Date: March 2007

Component Framework Specification

OMG Available Specification
formal/07-03-04

The PIM and PSM for Software Radio Components Specification (formal/07-03-01) is physically partitioned into 5 volumes:

Communication Channel and Equipment (formal/07-03-02)
Component Document Type Definitions (formal/07-03-03)
Component Framework (formal/07-03-04)
Common and Data Link Layer Facilities (formal/07-03-05)
POSIX Profiles (formal/07-03-06)

OBJECT MANAGEMENT GROUP

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are

brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

DISCLAIMER OF WARRANTY

RESTRICTED RIGHTS LEGEND

TRADEMARKS

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are

implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (*http://www.omg.org/technology/agreement.htm*).

Table of Contents

Preface v

1 Scope 1

2 Conformance 1

2.1 Conformance by a Model of a Specific Application 1
2.2 Conformance by a Tool 1
2.2.1 Definition of Terms for Discussion of Tool Conformance 1
2.2.2 Categories of Tool Conformance 2
2.3 Conformance on the part of a Component Framework PSM 2

3 References 3

3.1 Normative References 3
3.1.1 UML and Profile Specifications 3
3.1.1.1 UML Language Specification 3
3.1.1.2 OCL Language Specification 3
3.1.1.3 UML Profile for CORBA Specification 3
3.1.1.4 UML Profile for Modeling QoS and FT Characteristics and Mechanisms Specification 3
3.1.1.5 MOF 2.0/XMI Mapping Specification 4
3.1.2 CORBA Core Specifications 4
3.1.2.1 CORBA Specification 4
3.1.2.2 Real-time CORBA Specification 4
3.1.2.3 CORBA/e Specification 4
3.1.3 UML Profile Models 4
3.1.3.1 UML Profile for Component Framework 4
3.2 Non-normative References 4
3.2.1 Domain XML Profile 4
3.2.1.1 Domain XML Profile Files 4
3.2.1.2 Component Document Type Definitions Specification 5
3.2.1.3 Properties SML Schema Definition 5
3.2.2 Component Framework IDL 5
3.2.3 Common and Data Link Layer Facilities Specification 5
3.2.4 UML Profile for Communication Channel and Equipment Specification . 5

4 Terms and Definitions 5

5 Symbols and abbreviated terms 8

6 Additional Information 8

6.1 Changes to Adopted OMG Specifications 8
6.2 Guide to this Specification 9

6.3 Acknowledgements 9
7 UML Profile for Component Framework 11
7.1 Application and Device Components 11
7.1.1 Base Types 13
7.1.1.1 Any 13
7.1.1.2 BooleanSequence 13
7.1.1.3 Character 13
7.1.1.4 CharSequence 14
7.1.1.5 Double 14
7.1.1.6 DoubleSequence 14
7.1.1.7 ErrorNumberType 14
7.1.1.8 Float 15
7.1.1.9 FloatSequence 15
7.1.1.10 InvalidFileName 15
7.1.1.11 InvalidObjectReference 15
7.1.1.12 Long 15
7.1.1.13 LongDouble 15
7.1.1.14 LongDoubleSequence 16
7.1.1.15 LongLong 16
7.1.1.16 LongLongSequence 16
7.1.1.17 LongSequence 16
7.1.1.18 NameVersionCharacteristic 16
7.1.1.19 Object 16
7.1.1.20 Octet 17
7.1.1.21 OctetSequence 17
7.1.1.22 ObjectReference 17
7.1.1.23 ObjectRefSequence 17
7.1.1.24 Properties 17
7.1.1.25 PropertyValue 17
7.1.1.26 Short 18
7.1.1.27 ShortSequence 18
7.1.1.28 StringSequence 18
7.1.1.29 SystemException 18
7.1.1.30 ULong 18
7.1.1.31 ULongLong 19
7.1.1.32 ULongLongSequence 19
7.1.1.33 ULongSequence 19
7.1.1.34 UShort 19
7.1.1.35 UShortSequence 19
7.1.1.36 WChar 19
7.1.1.37 WCharSequence 20
7.1.1.38 WString 20
7.1.1.39 WStringSequence 20
7.1.1.40 TimeType 20
7.1.2 Literal Specifications 20
7.1.2.1 LiteralCharacter 21
7.1.2.2 LiteralDouble 21
7.1.2.3 LiteralFloat 21
7.1.2.4 LiteralLongDouble 21
7.1.2.5 LiteralWChar 22
7.1.2.6 LiteralWString 22
7.1.3 Properties 22

7.1.3.1 CapacityProperty 24
7.1.3.2 CharacteristicProperty 25
7.1.3.3 CharacteristicSelectionProperty 26
7.1.3.4 CharacteristicSetProperty 26
7.1.3.5 ConfigureProperty 27
7.1.3.6 EnumerationProperty 28
7.1.3.7 ExecutableProperty 28
7.1.3.8 InputValueProperty 29
7.1.3.9 Property 29
7.1.3.10 QueryProperty 29
7.1.3.11 ResultValueProperty 30
7.1.3.12 ServiceProperty 30
7.1.3.13 SimpleProperty 31
7.1.3.14 StructProperty 32
7.1.3.15 TestDefProperty 32
7.1.3.16 TestProperty 33
7.1.4 Interface and Port Stereotypes 33
7.1.4.1 RequiresPort 34
7.1.4.2 StreamPort 34
7.1.5 Resource Components 35
7.1.5.1 Resource Components Interfaces 35
7.1.5.2 Resource Components Stereotypes 44
7.1.6 Device Components 47
7.1.6.1 Device Component Interfaces 48
7.1.6.2 Device Component Stereotypes 54
7.1.7 Application Components 60
7.1.7.1 Application Components Types 60
7.1.7.2 Application Components Stereotypes 60
7.2 Infrastructure 70
7.2.1 Services 70
7.2.1.1 Services Interfaces 70
7.2.1.2 Services Stereotypes 74
7.2.1.3 File Services 78
7.2.2 Platform Management 87
7.2.2.1 Domain Management 87
7.2.2.2 Device Management 96
7.2.2.3 Domain Event Channels 101
7.2.3 Deployment 103
7.2.3.1 Artifacts 103
7.2.3.2 Applications Deployment 112
8 Platform Specific Model (PSM) 123
A Software Radio Reference Sheet 127

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

This specification responds to the requirements set by “Request for Proposals for a Platform Independent Model (PIM) and CORBA Platform Specific Model (PSM)” (swradio/02-06-02) of radio infrastructure facilities that can be utilized in developing applications such as waveforms, which promotes the portability of applications across platforms such as Software Defined Radios (SDR).

The Component Framework specification is physically partitioned into two major chapters: UML Profile for Component Framework and PSM for CORBA IDL and XML. UML Profile for Component Framework defines a language for modeling application, service, and domain management components by extending the UML language. The profile is specified independently from the underlying middleware technology and is applicable for other domains besides SDR.

This specification also provides a mechanism for transforming the elements of the profile model into the platform specific model for CORBA IDL and XML. This mapping definition is given in the PSM (Chapter 8).

Finally, the specification provides different compliance points depending on the role the implementer of this specification plays. Those different roles and respective partitioning of this document is given in the Conformance (Chapter 2).

2 Conformance

There are two kinds of conformance with respect to the profile: conformance on the part of a model of a specific application and conformance on the part of an MDA tool.

2.1 Conformance by a Model of a Specific Application

A UML model of a specific application either conforms to the profile or it does not. There are no categories of this kind of conformance. Such a UML model conforms to the profile if it satisfies all constraints imposed by the profile package.

2.2 Conformance by a Tool

2.2.1 Definition of Terms for Discussion of Tool Conformance

To support the discussion of conformance by a MDA tool, we define two terms: “identified subset of UML 2.0” and “all constructs defined by the profile.” The identified subset of UML 2.0 for the profile is the set of packages contained in the UML 2.0 Superstructure specification Part 1 (Structure). Part 1 includes the following packages and the transitive closure of all packages contained by these packages and of all packages upon which these packages depend:

- Classes
- Composite Structures
- Components
- Deployments

Hereafter we sometimes use the abbreviated term identified subset to refer to the identified subset of UML 2.0. The term all constructs defined by the profile is defined to mean all constructs that are part of the package's identified subset of UML 2.0, plus all extensions to that subset that the profile defines. Thus this term includes UML constructs that are part of the identified subset but that are not extended by the profile.

2.2.2 Categories of Tool Conformance

A tool is considered to be a conformant simple modeling tool for the profile if it does both of the following:

- Supports expression of all constructs defined by the profile, via UML 2.0 notation.
- Supports the UML 2.0 XMI exchange mechanism for the identified subset and for UML 2.0 profiles.

A tool is considered to be a conformant CORBA/XML-based forward engineering tool for the profile if it does the following:

- Supports the PIM-to-PSM Mapping defined in Chapter 8.
- Produces applications PSMs that are conformant to the behavior defined in the PIM.

Alternately, if a tool only produces an application skeleton, the skeleton must not make it impossible for a full application based on the skeleton to qualify as a conformant application; in other words, the skeleton must be able to form the basis of a conformant application.

A forward engineering tool that targets a platform technology other than CORBA/XML can legitimately claim a degree of conformance to the profile and PIM derived from the Profile if it conforms to the PIM-to-PSM Mapping and produces applications PSMs that are conformant applications to the behavior defined in the PIM, or produces application skeletons that can form the basis of conformant applications. In practice this requires the definition of an alternate PIM-PSM mapping.

A forward engineering tool of this nature for the platform "X" is considered to be a conformant X-Based forward engineering tool for the profile.

2.3 Conformance on the part of a Component Framework PSM

The interfaces and components as defined in Section 7 of this specification are not required to be used for a given platform or application. A platform or application uses the interfaces and component definitions that meet their needs. Conformance is at the level of usage as follows:

- A PSM implementation (no matter what language) of an interface defined in this specification needs to be conformant to the interface definition as described in the specification.
- A PSM implementation (no matter what language) of a component defined in this specification needs to be conformant to the component definition (ports, interfaces realized, properties, etc.) as described in the specification.

A component is considered to be a conformant for Component Framework CORBA/XML platform if it does all of the following:

- Implements the CORBA interfaces that the component PSM defines
- Implements the XML serialization formats that the component PSM defines
- Implements the semantics that the component PIM defines

Note that the component PIM essentially defines the semantics for the CORBA interfaces and XML serialization formats. The semantics for a CORBA interface defined in the component PSM are defined by the semantics of the corresponding element(s) in the component PIM. It is possible to deduce the corresponding elements in the PIM for such a CORBA interface by reversing the PIM-PSM Mapping.

3 References

3.1 Normative References

3.1.1 UML and Profile Specifications

3.1.1.1 UML Language Specification

Unified Modeling Language (UML) Superstructure Specification, Version 2.1.1
Formal OMG Specification, document number: formal/07-02-03
The Object Management Group, February 2007
[http://www.omg.org]

Unified Modeling Language (UML) Infrastructure Specification, Version 2.1.1
Formal OMG Specification, document number: formal/07-02-04
The Object Management Group, February 2007
[http://www.omg.org]

3.1.1.2 OCL Language Specification

Object Constraint Language (OCL) Specification, Version 2.0
Formal OMG Specification, document number: formal/2006-05-01
The Object Management Group, May 2006
[http://www.omg.org]

3.1.1.3 UML Profile for CORBA Specification

UML Profile for CORBA Specification, Version 1.0
Formal OMG Specification, document number: formal/2002-04-01
The Object Management Group, April 2002
[http://www.omg.org]

3.1.1.4 UML Profile for Modeling QoS and FT Characteristics and Mechanisms Specification

UML Profile for Modeling QoS and FT Characteristics and Mechanisms, Version 1.0
Formal OMG Specification, document number: formal/06-05-02
The Object Management Group, May 2006
[http://www.omg.org]

3.1.1.5 MOF 2.0/XMI Mapping Specification

Meta Object Facility (MOF) 2.0 XMI Mapping Specification, Version 2.1
Formal OMG Specification, document number: formal/05-09-01
The Object Management Group, September 2005
[http://www.omg.org]

3.1.2 CORBA Core Specifications

3.1.2.1 CORBA Specification

Common Object Request Broker (CORBA/IIOP), Version 3.0.3
Formal OMG Specification, document number: formal/2004-03-01
The Object Management Group, March 2004
[http://www.omg.org]

3.1.2.2 Real-time CORBA Specification

Real-time - CORBA Specification, Version 1.2
Formal OMG Specification, document number: formal/2005-01-04
The Object Management Group, January 2005
[http://www.omg.org]

3.1.2.3 CORBA/e Specification

CORBA/e Specification
Draft Adopted OMG Specification, document number: ptc/06-05-01
The Object Management Group, May 2006
[http://www.omg.org]

3.1.3 UML Profile Models

3.1.3.1 UML Profile for Component Framework

UML Profile for Component Framework XMI File
Formal OMG document number: dtc/2006-04-09
The Object Management Group, December 2006
[http://www.omg.org]

3.2 Non-normative References

3.2.1 Domain XML Profile

3.2.1.1 Domain XML Profile Files

Domain DTDs XML Files
Formal OMG document number:dtc/2006-04-13
The Object Management Group, December 2006
[http://www.omg.org]

3.2.1.2 Component Document Type Definitions Specification

Component Document Type Definitions Specification
Formal OMG document number:dtc/2006-04-07
The Object Management Group, December 2006
[http://www.omg.org]

3.2.1.3 Properties SML Schema Definition

Properties XML Schema File
Formal OMG document number:dtc/2006-04-12
The Object Management Group, December 2006
[http://www.omg.org]

3.2.2 Component Framework IDL

Component Framework IDL Files
Formal OMG document number:dtc/2006-04-16
The Object Management Group, December 2006
[http://www.omg.org]

3.2.3 Common and Data Link Layer Facilities Specification

Common and Data Link Layer Facilities Specification, Version 1.0
Formal OMG document number: formal/07-03-05
The Object Management Group, March 2007
[http://www.omg.org]

3.2.4 UML Profile for Communication Channel and Equipment Specification

Communication Channel and Equipment Specification, Version 1.0
Formal OMG document number: formal/07-03-02
The Object Management Group, March 2007
[http://www.omg.org]

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

Common Object Request Broker Architecture (CORBA)

An OMG distributed computing platform specification that is independent of implementation languages.

Component

A component can always be considered an autonomous unit within a system or subsystem. It has one or more ports, and its internals are hidden and inaccessible other than as provided by its interfaces. A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component

exposes a set of ports that define the component specification in terms of provided and required interfaces. As such, a component serves as a type, whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics).

Facility

The realization of certain functionality through a set of well defined interfaces.

Interface Definition Language (IDL)

An OMG and ISO standard language for specifying interfaces and associated data structures.

Logical Device

A software component that is an abstraction of a hardware device it represents.

Mapping

The Specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel.

Metadata

The Data that represents models. For example, a UML model; a CORBA object model expressed in IDL; and a relational database schema expressed using CWM.

Metamodel

A model of models.

Meta Object Facility (MOF)

An OMG standard, closely related to UML, that enables metadata management and language definition.

Model

A formal specification of the function, structure and/or behavior of an application or system.

Model Driven Architecture (MDA)

An approach to IT system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform.

Platform

A set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented.

Platform Independent Model (PIM)

A model of a subsystem that contains no information specific to the platform, or the technology that is used to realize it.

Platform Specific Model (PSM)

A model of a subsystem that includes information about the specific technology that is used in the realization of it on a specific platform, and hence possibly contains elements that are specific to the platform.

Request for Proposal (RFP)

A document requesting OMG members to submit proposals to the OMG's Technology Committee. Such proposals must be received by a certain deadline and are evaluated by the issuing task force.

Service

A set of functionality with common characteristics.

Unified Modeling Language (UML)

An OMG standard language for specifying the structure and behavior of systems. The standard defines an abstract syntax and a graphical concrete syntax.

UML Profile

A standardized set of extensions and constraints that tailors UML to particular use.

5 Symbols and abbreviated terms

Abbreviation	Definition
API	Application Program Interface
BIT	Built-In Test
CORBA	Common Object Request Broker Architecture
HCI	Human-Computer Interface
ID	Identification, Identifier
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
ISO	International Standards Organization
N/A	Not Applicable
OE	Operating Environment
OMG	Object Management Group
ORB	Object Request Broker
OS	Operating System
OSI	Open System Interconnection
PIM	Platform Independent Model
POSIX	Portable Operating System Interface
PSM	Platform Specific Model
SDR	Software Defined Radio
UML	Unified Modeling Language
UUID	Universally Unique Identifier
XML	eXtensible Markup Language

6 Additional Information

6.1 Changes to Adopted OMG Specifications

The specifications contained in this document require no changes to adopted OMG specifications.

6.2 Guide to this Specification

This specification consists of two major parts, contained in the following chapters 7 and 8.

Chapter 7 defines the modeling language used in this specification in form of a UML profile for a generic component framework.

Chapter 8 contains a description of the mapping process from the Platform Independent Model (PIM) to a Platform Specific Model (PSM).

The normative UML profile referenced in Section 3.1.1 is used to generate the class diagrams shown throughout this specification.

6.3 Acknowledgements

The following organizations (listed in alphabetical order) contributed to this specification:

- BAE Systems
- The Boeing Company
- Blue Collar Objects
- Carleton University
- Communications Research Center Canada
- David Frankel Consulting
- École de Technologie Supérieure
- General Dynamics Decision Systems
- Harris
- ISR Technologies
- ITT Aerospace/Communications Division
- L-3 Communications Corporation
- Mercury Computer Systems
- The MITRE Corporation
- Mobile Smarts
- Northrup Grumman
- PrismTech
- Raytheon Corporation
- Rockwell Collins
- SCA Technica
- Space Coast Communication Systems
- Spectrum Signal Processing
- THALES

- Virginia Tech University
- Zeligsoft
- 88soultions

7 UML Profile for Component Framework

This non-normative section defines the UML Profile for Component Framework. This profile is an integral part of the “PIM and PSM for Software Radio Components.” The set of stereotypes defined in this profile constitutes the core language for the definition of the SWRadio PIM and PSM. The current UML Profile for Component Framework extends the UML 2.0 meta-language, with emphasis on extensions to the Components package and Deployment package of UML 2.0.

The goal of the UML Profile for Component Framework is to enable the development of UML tools to support the development of applications and systems. The objectives are not only to facilitate the modeling of applications and systems, but also to enable the automatic generation of descriptor files (e.g., XML descriptor files) and code (or code skeletons) from UML models.

To address the issues of the different actors involved in product developments, the current profile has been developed with two main viewpoints in mind: the viewpoint of application and device developers and the viewpoint of infrastructure/middleware providers. These two viewpoints define distinct sets of concepts (and stereotypes) that are required in different contexts.

To be consistent with the two viewpoints introduced above, the UML Profile for Component Framework is partitioned in two main packages: the Applications and Devices package and the Infrastructure package. Each package defines the set of concepts and UML stereotypes required to perform a specific role in the development of a product.

The Applications and Devices package defines the set of concepts that are required to develop applications and devices. This package mainly contains a set of stereotypes that extends the UML 2.0 meta-classes Component and Interface. This set of stereotypes includes Resource and Device components.

The Infrastructure package defines the concepts that are required to deploy services and applications within a platform infrastructure, and to manage the domain, services, and devices. This package mainly contains a set of stereotypes that extends the UML 2.0 meta-classes Component, Artifact, and Interface. This set of stereotypes includes DomainManager, DeviceManager, ApplicationManager, and ApplicationFactory components.

The format for stereotype names (e.g., resourcecomponent) in the profile is all lower case letters. In this specification the stereotype names are shown with mixture of upper and lower case letters, where each word starts with an upper case letter (e.g., ResourceComponent).

7.1 Application and Device Components

The Applications and Devices package provides a set of component and interface stereotype definitions that are used for the development of applications, logical devices, and components. For application developers all sections are applicable except for Device Components section and for device developers all sections are applicable except for Application Components section. Figure 7.1 depicts the relationships among the packages within the Application and Device Components, which are as follows:

- Base Types - defines basic types for applications, logical devices, and component definitions.
- Properties - defines stereotypes for configure, query, testable, service artifact (capability and capacity) and executable properties for components and executable code artifacts.
- Interface & Port Stereotypes - defines stereotypes for interfaces and components.

- Resource Components - defines stereotypes for the interfaces and components for the ResourceComponent, which is the basic component type for application and device components.
- Device Components - defines stereotypes for the logical device components that represent devices that components are deployed on or use within a platform.
- Application Components - defines stereotypes for the ResourceComponents for applications.

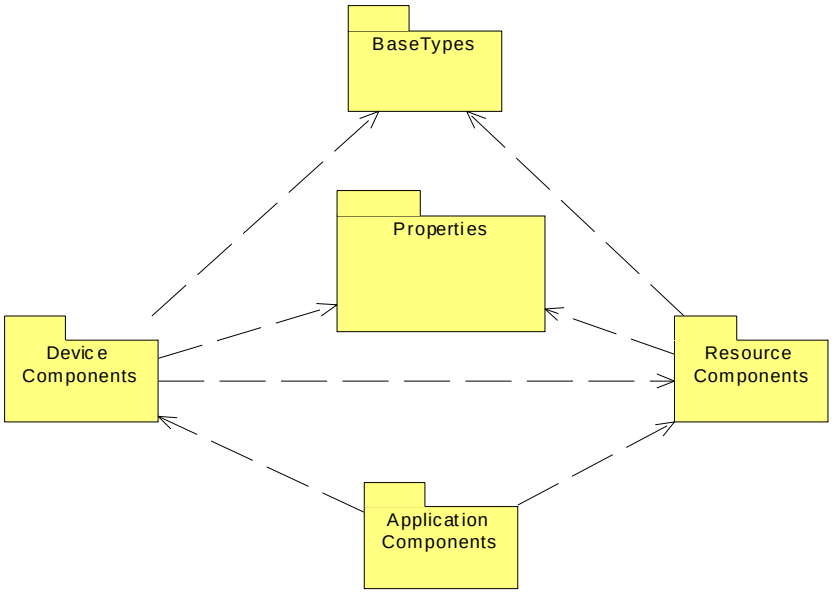

Figure 7.1 - Applications and Devices Overview

7.1.1 Base Types

The Base Types defines the basic types and exceptions used by multiple components as shown in Figure 7.2.

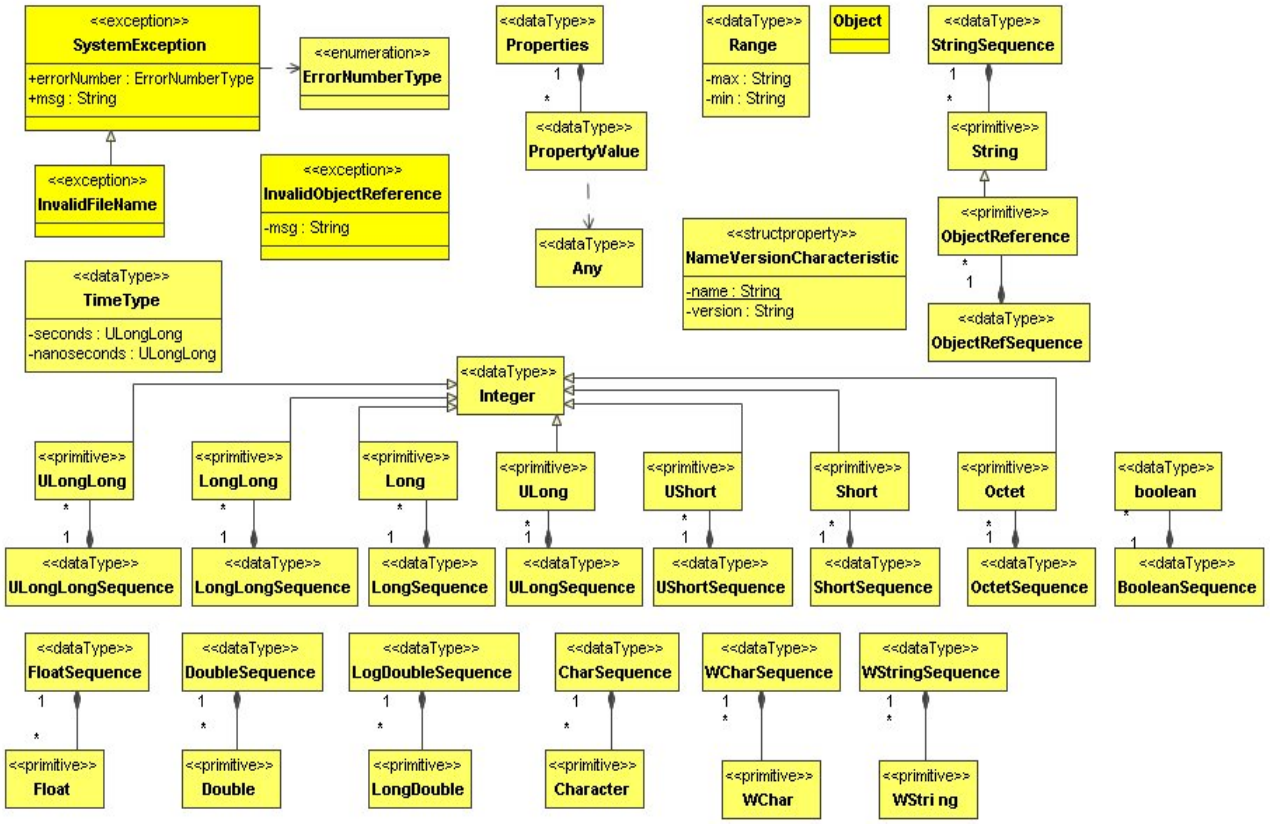

Figure 7.2 - Base Types Overview

7.1.1.1 Any

Description

The Any data type represents a type that can take on any value.

7.1.1.2 BooleanSequence

Description

The BooleanSequence data type is an unbounded sequence of Boolean(s).

7.1.1.3 Character

Description

The Character primitive type is an 8-bit quantity that encodes a single-byte character from any byte-oriented code set.

Constraints

The Character value is a LiteralCharacter.

7.1.1.4 CharSequence

Description

The CharSequence data type is an unbounded sequence of Character(s).

7.1.1.5 Double

Description

The Double primitive type is an IEEE double-precision floating point number. See IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, for a detailed specification.

Constraints

The Double value is a LiteralDouble.

7.1.1.6 DoubleSequence

Description

The DoubleSequence data type is an unbounded sequence of Double(s).

7.1.1.7 ErrorNumberType

Description

The ErrorNumberType enumeration defines error number information used in various exceptions.

Attributes

Those enumeration literal names starting with “CF_E” map to the POSIX definitions (starting with “E”) that can be found in IEEE Std. 1003.1 1996 Edition. CF_NOTSET is not defined in the POSIX specification. CF_NOTSET is a specific value that is applicable for any exception when the method specific or standard POSIX error values are not appropriate.

Enumeration Literal names are:

CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR, CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV

7.1.1.8 Float

Description

The Float primitive type is an IEEE single-precision floating point number. See IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, for a detailed specification.

Constraints

The Float value shall be a LiteralFloat that is an IEEE single-precision floating point number.

7.1.1.9 FloatSequence

Description

The FloatSequence data type is an unbounded sequence of Float(s).

7.1.1.10 InvalidFileName

Description

The InvalidFileName <<exception>>, type of SystemException, indicates an invalid file name. The errorNumber attribute indicates the type of error (e.g., CF_ENAMETOOLONG). The String msg attribute provides information describing why the filename was invalid.

7.1.1.11 InvalidObjectReference

Description

The InvalidObjectReference <<exception>> indicates an invalid object reference error.

Attributes

- `msg: String`
  A msg attribute is supplied for further information on the exception being raised.

7.1.1.12 Long

Description

The Long primitive type, a specialization of Integer primitive type, is a signed integer with a range of -2^{31}.. $2^{31} - 1$.

Constraints

The Long value shall be a LiteralInteger with a range of -2^{31}.. $2^{31} - 1$.

7.1.1.13 LongDouble

Description

The LongDouble primitive type is an IEEE double-extended floating-point number. See IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, for a detailed specification.

Constraints

The LongDouble value shall be a LiteralLongDouble that is an IEEE double-extended floating-point number.

7.1.1.14 LongDoubleSequence

Description

The LongDoubleSequence data type is an unbounded sequence of LongDouble(s).

7.1.1.15 LongLong

Description

The LongLong primitive type, a specialization of Integer primitive type, is a signed integer with a range of -2^{63} .. $2^{63} - 1$.

Constraints

The LongLong value shall be a LiteralInteger with a range of -2^{63} .. $2^{63} - 1$.

7.1.1.16 LongLongSequence

Description

The LongLongSequence data type is an unbounded sequence of LongLong(s).

7.1.1.17 LongSequence

Description

The LongSequence data type is an unbounded sequence of Long(s).

7.1.1.18 NameVersionCharacteristic

Description

The NameVersionCharacteristic, a StructProperty type, defines a characteristic that consists of a name and a version pair, which can be used for runtime, library, and os characteristic definitions.

Attributes

- `name: String`
  The name attribute identifies the element.
- `version: String`
  The version attribute identifies the version of the element.

7.1.1.19 Object

Description

The Object represents a type that is a reference to an object instance.

7.1.1.20 Octet

Description

The Octet primitive type, a specialization of Integer primitive type, is an unsigned integer with a range of 0..255.

Constraints

The Octet value shall be a LiteralInteger with a range of 0..255.

7.1.1.21 OctetSequence

Description

The OctetSequence data type is an unbounded sequence of octets.

7.1.1.22 ObjectReference

Description

The ObjectReference primitive type, a specialization of String primitive type, is a stringified object reference of an object.

Constraints

The ObjectReference value is a LiteralString.

7.1.1.23 ObjectRefSequence

Description

The ObjectRefSequence data type is an unbounded sequence of ObjectReference(s).

7.1.1.24 Properties

Description

The Properties, as shown in Figure 7.2, is an unbounded sequence of PropertyValue(s), which is used in defining a sequence of id and value pairs.

7.1.1.25 PropertyValue

Description

The PropertyValue is used to hold a property’s value.

Attributes

- `id: String`
  The id attribute identifies a specific property of the component.
- `value: Any`
  The value attribute contains the property's value.

Constraints

The value attribute shall be either a primitive type (e.g., String, ULong, etc.), primitive sequence (e.g., StringSequence, ULongSequence, etc.) or a Properties type.

7.1.1.26 Short

Description

The Short primitive type, a specialization of Integer primitive type, is a signed integer with a range of -2^{15}.. $2^{15} - 1$.

Constraints

The Short value shall be a LiteralInteger with a range of -2^{15}.. $2^{15} - 1$.

7.1.1.27 ShortSequence

Description

The ShortSequence data type is an unbounded sequence of Short(s).

7.1.1.28 StringSequence

Description

The StringSequence data type is an unbounded sequence of Strings.

7.1.1.29 SystemException

Description

The SystemException exception, as shown in Figure 7.2, denotes a type that is used when an exception is raised by an operation.

Attributes

- `errorNumber: ErrorNumberType`
  The errorNumber indicates the type of system error.
- `msg: String`
  The message attribute is used to add additional information on the error that occurred.

7.1.1.30 ULong

Description

The ULong primitive type, a specialization of Integer primitive type, is an unsigned integer with a range of 0.. $2^{32} - 1$.

Constraints

The ULong value shall be a LiteralInteger with a range of 0.. $2^{32} - 1$.

7.1.1.31 ULongLong

Description

The ULongLong primitive type, a specialization of Integer primitive type, is an unsigned integer with a range of 0.. (2^{64} - 1).

Constraints

The ULongLong primitive type, a specialization of Integer primitive type, is an unsigned integer with a range of 0.. (2^{64} - 1).

7.1.1.32 ULongLongSequence

Description

The ULongLongSequence data type is an unbounded sequence of ULongLong(s).

7.1.1.33 ULongSequence

Description

The ULongSequence data type is an unbounded sequence of ULong(s).

7.1.1.34 UShort

Description

The UShort primitive type, a specialization of Integer primitive type, is an unsigned integer with a range of 0.. $2^{16} – 1$.

Constraints

The UShort value shall be a LiteralInteger with a range of 0.. $2^{16} – 1$.

7.1.1.35 UShortSequence

Description

The UShortSequence data type is an unbounded sequence of UShort(s).

7.1.1.36 WChar

Description

The WChar primitive type represents a wide character that can be used for any character set.

Constraints

The WChar value shall be a LiteralWChar.

7.1.1.37 WCharSequence

Description

The WCharSequence data type is an unbounded sequence of WChar(s).

7.1.1.38 WString

Description

The WString primitive type represents a wide character sting that can be used for any character set.

Constraints

The WString value is a LiteralWString.

7.1.1.39 WStringSequence

Description

The WStringSequence data type is an unbounded sequence of WString(s).

7.1.1.40 TimeType

Description

The TimeType, as shown in Figure 7.2, denotes a type that represents time.

Attributes

- `seconds: ULongLong`
  Seconds
- `nanoseconds: ULongLong`
  Nanoseconds

Semantics

The TimeType attribute is used to define time. Seconds in the future when an event should occur or in the past when an event has occurred. Seconds field is the seconds that have occurred since last synchronization epoch. (Epoch method will be defined by instantiating API.) In most cases, the epoch will occur one time when the box is initialized. Nanosecond offset from the seconds field (described above) when a future event will occur or past event has occurred.

7.1.2 Literal Specifications

A literal specification is an abstract specialization of ValueSpecification that identifies a literal constant being modeled. The literal specifications identified in this section are extensions of the UML LiteralSpecification metaclass.

7.1.2.1 LiteralCharacter

Description

A literal character, an extension of LiteralSpecification, contains a Character-valued attribute.

Attributes

- value : Character

Semantics

A LiteralCharacter specifies a constant Character value.

7.1.2.2 LiteralDouble

Description

A literal double, an extension of LiteralSpecification, contains a Double-valued attribute.

Attributes

- value : Double

Semantics

A LiteralDouble specifies a constant Double value.

7.1.2.3 LiteralFloat

Description

A literal float, an extension of LiteralSpecification, contains a Float-valued attribute.

Attributes

- value : Float

Semantics

A LiteralFloat specifies a constant Float value.

7.1.2.4 LiteralLongDouble

Description

A literal long double, an extension of LiteralSpecification, contains a LongDouble-valued attribute.

Attributes

- value : LongDouble

Semantics

A LiteralLongDouble specifies a constant LongDouble value.

7.1.2.5 LiteralWChar

Description

A literal wide character, an extension of LiteralSpecification, contains a WChar-valued attribute.

Attributes

- value : WChar

Semantics

A LiteralWChar specifies a constant wide character value.

7.1.2.6 LiteralWString

Description

A literal wide string, an extension of LiteralSpecification, contains a WString-valued attribute.

Attributes

- value : WString

Semantics

A LiteralWString specifies a constant wide string value.

7.1.3 Properties

This section defines the property stereotypes for components. A property is a named value denoting an attribute of a class. The property types contained in this package are configure, query, simple, test, structure, and service. All properties are based upon primitive data type values (e.g., Character, ULong, string, etc.). The reason for this is two fold: 1) these primitive types are supported by distributed component middleware, and 2) the primitive types allow for a generic mechanism to be built such as deployment, component interaction, and Human Computer Interface (HCI). All the values of a sequence type (simple or structure sequence) are based upon the same property definition, in order to simplify processing within an embedded environment. Simple, Structure, and Sequence properties can be used for configuring and/or querying a component's properties. Test properties are testable properties for a component. There are six subclasses of a SimpleProperty which are CapacityProperty, CharacteristicProperty, CharacteristicSelectionProperty, InputValueProperty, ResultValueProperty, and ExecutableProperty. The ServiceProperty is used to describe the characteristic capabilities and capacities of a Service. The ExecutableProperty is used to describe the executable parameters for an executable implementation (e.g., process, thread). The details of each property type are described in the following subsections.

Table 7.1 - Properties Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
CapacityProperty	Property	ServiceProperty, SimpleProperty		See constraints in section below	Represents capacity property.
CharacteristicProperty	Property	ServiceProperty, SimpleProperty		See constraints in section below	Represents characteristic property.
CharacteristicSelection Property	Property	ServiceProperty, SimpleProperty		See constraints in section below	Represents characteristic property that is a simple list.
CharacteristicSetProperty	Property	ServiceProperty, Property		See constraints in section below	Represents a set of characteristic properties that are of the same classification.
ConfigureProperty	Property	Property	stepSize	See constraints in section below	Represents a configurable and queryable property.
EnumerationProperty	Data Type	N/A		See constraints in section below	Represents a enumeration property type where the enumeration literals have an integer value.
ExecutableProperty	Property	SimpleProperty	queryable	See constraints in section below	Represents an executable parameter property.
InputValueProperty	Property	SimpleProperty			Represents an input value parameter property for a test.
<<abstract>>Property	Property	N/A	ID, label, maxLatency, range, units	See constraints in section below	Represents the common attributes for all property types.
QueryProperty	Property	Property		See constraints in section below	Represents a queryable property.
ResultValueProperty	Property	SimpleProperty			Represents a result value for a test.
<<abstract>>Service Property	Property	N/A	capabilityModel, locallyManaged	See constraints in section below	Represents characteristic or capacity property for a service.
SimpleProperty	Property	Property		See constraints in section below	Represents a Property that contains a primitive value type.

Table 7.1 - Properties Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
StructProperty	Data Type	N/A		See constraints in section below.	Represents a struct of SimpleProperties.
TestDefProperty	Data Type	N/A		See Constraints in section below.	Describes a test definition type and its associated inputs and expected results.
TestProperty	Property	Property		See constraints in section below.	Describes a test property.

7.1.3.1 CapacityProperty

Description

The CapacityProperty as shown in Figure 7.3 is a type of ServiceProperty and SimpleProperty that defines a managed or unmanaged dynamic capacity for a ServiceComponent.

Constraints

- The CapacityProperty's locallyManaged attribute default value shall be True. The corresponding OCL is as follows:
  context CapacityProperty::locallyManaged:Boolean init: true
- Valid values for the CapacityProperty's capabilityModel attribute value shall be: "counter" and "quantity." The corresponding OCL is as follows:
  context CapacityProperty inv validcapabilitymodel: self.capabilityModel = 'counter' or self.capabilityModel = 'quantity'
- CapacityProperty shall have an initial value of 'quantity.' The corresponding OCL is as follows:
  context CapacityProperty::capabilityModel:String init: 'quantity'

Semantics

CapacityProperty's type is usually a numeric type (e.g., ULong, Float, etc.).

The meanings of the CapacityProperty's capabilityModel attribute values shall be:

- "counter" - This CapacityModel has a capacity of a counter. The capacity value is decremented by one until zero during allocation and incremented by one during deallocation. Allocation fails if the counter is at zero. Example: a sound card with 4 output channels.
- "quantity" - This CapacityModel has a certain capacity that can be consumed. The value of the deployment requirement value is subtracted from the capacity during allocation and added to the capacity during deallocation. The allocation is successful if the capacity has a value that equals or exceeds the value of the deployment requirement. Example: memory size.

Other capabilityModel attribute values can be specified. These other specified capabilityModel attribute values may be managed at the ManagedServiceComponent level or at an ApplicationFactory level.

7.1.3.2 CharacteristicProperty

Description

The CharacteristicProperty is a type of ServiceProperty and SimpleProperty that defines a static characteristic for a ServiceComponent.

Constraints

- The CharacteristicProperty's locallyManaged attribute default value shall be False. The corresponding OCL is as follows:
  context CharacteristicProperty::locallyManaged:Boolean init: false
- Valid values for the CharacteristicProperty's capabilityModel attribute value shall be: "eq," "ne," "le," "ge," "lt," "gt," "maximum," or "minimum." The corresponding OCL is as follows:
  context CharacteristicProperty inv validcapabilitymodel: self.capabilityModel in Set { 'eq,' 'ne,' 'le,' 'ge,' 'lt,' 'gt,' 'maximum,' 'minimum' }
- CharacteristicProperty shall have an initial value of 'eq.' The corresponding OCL is as follows:
  context CharacteristicProperty::capabilityModel:String init: 'eq'

Semantics

The meanings of the CharacteristicProperty's capabilityModel attribute values shall be:

- "eq" - is an equality comparison between the CharacteristicProperty's value attribute and a deployment requirement (Infrastructure:: Deployment:: Artifacts::BasicDeploymentRequirement). They are equal if the CharacteristicProperty's value equals the deployment requirement. If they are equal, then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.
- "ne" - is a not equal comparison between the CharacteristicProperty's value attribute and a deployment requirement. They are not equal if CharacteristicProperty's value does not equal the deployment requirement. If they are not equal, then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.
- "le" - is a less than or equal comparison between the CharacteristicProperty's value attribute and a deployment requirement. If the deployment requirement is less than or equal to the CharacteristicProperty, then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.
- "ge" - is a greater than or equal comparison between the CharacteristicProperty's value attribute and a deployment requirement. If the deployment requirement is greater than or equal to the CharacteristicProperty, then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement deployment requirement otherwise it cannot satisfy the deployment requirement.
- "lt" - is a less than comparison between the CharacteristicProperty's value attribute and a deployment requirement. If the deployment requirement is less than the CharacteristicProperty, then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.
- "gt" - is a greater than comparison between the CharacteristicProperty's value attribute and a deployment requirement. If the deployment requirement is greater than the CharacteristicProperty, then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.
- "Minimum" - behaves as "ge"
- "Maximum" - behaves as "le"

Other capabilityModel attribute values can be specified. These other specified capabilityModel attribute values may be managed at the ServiceComponent level or at an ApplicationFactory level.

7.1.3.3 CharacteristicSelectionProperty

Description

The CharacteristicSelectionProperty is a type of ServiceProperty and SimpleProperty that defines a static characteristic for a ServiceComponent. The property contains a list of primitive characteristic values that can be selected from for a match.

Constraints

- The CharacteristicSelectionProperty's locallyManaged attribute default value shall be False. The corresponding OCL is as follows:
  context CharacteristicSelectionProperty::locallyManaged:Boolean init: false
- Valid values for the CharacteristicSelectionProperty's capabilityModel attribute value shall be: "selection." The corresponding OCL is as follows:
  context CharacteristicSelectionProperty inv validcapabilitymodel: self.capabilityModel = 'selection'
- CharacteristicSelectionProperty shall have an initial value of 'selection.' The corresponding OCL is as follows:
  context CharacteristicSelectionProperty::capabilityModel:String init: 'selection'

Semantics

The meanings of the CharacteristicSelectionProperty's capabilityModel attribute values are:

- "selection" - is an equality/match comparison between any CharacteristicSelectionProperty's value attribute and a BasicDeploymentRequirement (Infrastructure:: Deployment:: Artifacts). They shall be equal/match if any SelectionCharacteristicProperty's value equals the deployment requirement. If they are equal/match, then the CharacteristicSelectionProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.

Other capabilityModel attribute values can be specified. These other specified capabilityModel attribute values may be managed at the ServiceComponent level or at an ApplicationFactory level.

7.1.3.4 CharacteristicSetProperty

Description

The CharacteristicSetProperty is a type of ServiceProperty and Property that defines a characteristic that defines the same set of characteristics for a ServiceComponent.

Constraints

- The CharacteristicSetProperty's locallyManaged attribute default value shall be False. The corresponding OCL is as follows:
  context CharacteristicSetProperty::locallyManaged:Boolean init: false
- Valid values for the CharacteristicSetProperty's capabilityModel attribute value shall be: "selection."
  The corresponding OCL is as follows:
  context CharacteristicSetProperty inv validcapabilitymodel: self.capabilityModel = 'selection'

- CharacteristicSetProperty shall have an initial value of ‘selection.’ The corresponding OCL is as follows:
  context CharacteristicSetProperty::capabilityModel:String init: ‘selection’
- Each CharacteristicSetProperty’s attribute type shall be a stereotyped as StructProperty.
  The corresponding OCL is as follows:
  context CharacteristicSetProperty inv: self.allAttributes()->collect(a | a.type.stereotype = ‘StructProperty’)
- Each CharacteristicSetProperty’s attribute type shall be the same type definition.
  The corresponding OCL is as follows:
  context CharacteristicSetProperty inv: self.allAttributes()->collect(a | a.type)->size() = 1

Semantics

The meanings of the CharacteristicSetProperty’s capabilityModel attribute values are:

- “selection” - is an equality/match comparison between any CharacteristicSetProperty’s value attribute and a DeploymentRequirementQualifier (Infrastructure:: Deployment:: Artifacts). They shall be equal/match if any CharacteristicSetProperty’s characteristic equals the deployment requirement. If they are equal/match, then the CharacteristicSetProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.

Other capabilityModel attribute values can be specified. These other specified capabilityModel attribute values may be managed at the ServiceComponent level or at an ApplicationFactory level.

7.1.3.5 ConfigureProperty

Description

The ConfigureProperty, a type of Property, indicates a configurable and queryable property. There are five types of ConfigureProperty, which are: primitive types, primitive sequence types, EnumerationProperty, StructProperty, and StructProperty sequences.

Attributes

- `stepSize: ULong [0..1]`
  The stepSize attribute represents the fact that some properties have discrete increments. An example is a tunable duplexer, which uses a stepper motor to adjust the tuned frequency.

Constraints

- ConfigureProperty isReadOnly attribute shall be false. The corresponding OCL is as follows:
  context ConfigureProperty inv validisreadonly: self.isReadOnly = false
- The type for ConfigureProperty shall be constrained to be primitive types (e.g., String, ULong, etc.), primitive sequence types (e.g., StringSequence, ULongSequence, etc.), EnumerationProperty, and StructProperty types.

Semantics

The ConfigureProperty defines properties associated with the PropertySet interface implementations. The properties supported by a component are described in a component’s descriptor.

7.1.3.6 EnumerationProperty

Description

The EnumerationProperty, an extension of Data Type, as shown in Table 7.1 defines an enumeration type where the enumeration literals have an integer value.

Constraints

- The EnumerationProperty attributes' type shall be of the same integer type (Octet, Short, UShort, Long, ULong, ULongLong, LongLong).
- Each EnumerationProperty's attribute name shall be unique within the EnumerationProperty. The corresponding OCL is as follows:
  context EnumerationProperty inv: self.allAttributes()->isUnique(a | a.name)
- Each EnumerationProperty's attribute values shall be unique within the EnumerationProperty.
  The corresponding OCL is as follows:
  context EnumerationProperty inv: self.allAttributes()->isUnique(a | a.value)
- Each EnumerationProperty's attribute value shall be in the range of the EnumerationProperty Type.

Semantics

The EnumerationProperty forms an enumeration type definition. EnumerationProperty is legal for integer type properties elements. The EnumerationProperty attributes are enumeration literals, which have an integer value. EnumerationProperty attribute values are implied; if not specified, the initial value is 0 and subsequent values are incremented by 1. This allows a configure, query, or characteristic property to be expressed as an enumeration with integer values.

7.1.3.7 ExecutableProperty

Description

The ExecutableProperty, a type of SimpleProperty as shown in Table 7.1, defines executable parameters for an ExecutableCode element such as a main process.

Attributes

- `Queryable : Boolean = True`
  The queryable attribute indicates whether or not the ExecutableProperty can be queried for its value. True means the property is queryable.

Constraints

- The ExecutableProperty's value shall be specified.

7.1.3.8 InputValueProperty

Description

The InputValueProperty, a type of SimpleProperty as shown in Table 7.1, provides the capability to define an input property for a TestDefProperty.

7.1.3.9 Property

Description

The abstract Property is an extension of the UML Property that provides the basic attributes for all properties definitions.

Attributes

- `ID : String [0..1]`
  The optional ID attribute is a string that represents the identifier for the property.
- `maxLatency: TimeType [0..1]`
  The maxLatency attribute represents the time needed by an attribute to achieve its proper operating status. An example is the gain of an amplifier. The maxLatency indicates the time that the amplifier needs before it attains its operating gain. The latency is measured from power-up.
- `range: Range [0..1]`
  The optional range attribute indicates the allowable min and max values for value attribute.
- `units: String [0..1]`
  The optional units attribute indicates the unit of measure for the value attribute.

Types and Exceptions

- `Range (min: String, max: String)`
  The Range type defines the min and max allowable values for a property. The String value represents a numerical value.

Constraints

- The ID attribute when specified shall have precedence over the name attribute for the identification of the Property.
- The value for the name attribute shall be case sensitive.

Semantics

The name or ID attribute is used for property identification. The name attribute is also used for display purposes. The ID is usually not human readable or meaningful such as a Universal Unique IDentifier (UUID) value or an integer.

7.1.3.10 QueryProperty

Description

The QueryProperty, a type of Property as shown in Table 7.1, provides the capability to define a read-only property that can be queried at run-time by a control command. It cannot be modified by a control command.

Constraints

- The type for QueryProperty shall be constrained to be primitive types (e.g., String, ULong, etc.), primitive sequence types (e.g., StringSequence, ULongSequence, etc.), EnumerationProperty, and StructProperty types.
- The QueryProperty isReadOnly attribute value shall be always set to true. The corresponding OCL is as follows: context QueryProperty inv validisreadonly: self.isReadOnly = true.

7.1.3.11 ResultValueProperty

Description

The ResultValueProperty, a type of SimpleProperty as shown in Table 7.1, provides the capability to define a result property for a TestDefProperty.

7.1.3.12 ServiceProperty

Description

The abstract ServiceProperty, as shown in Figure 7.3, defines capability and/or capacity characteristics for a ServiceComponent.

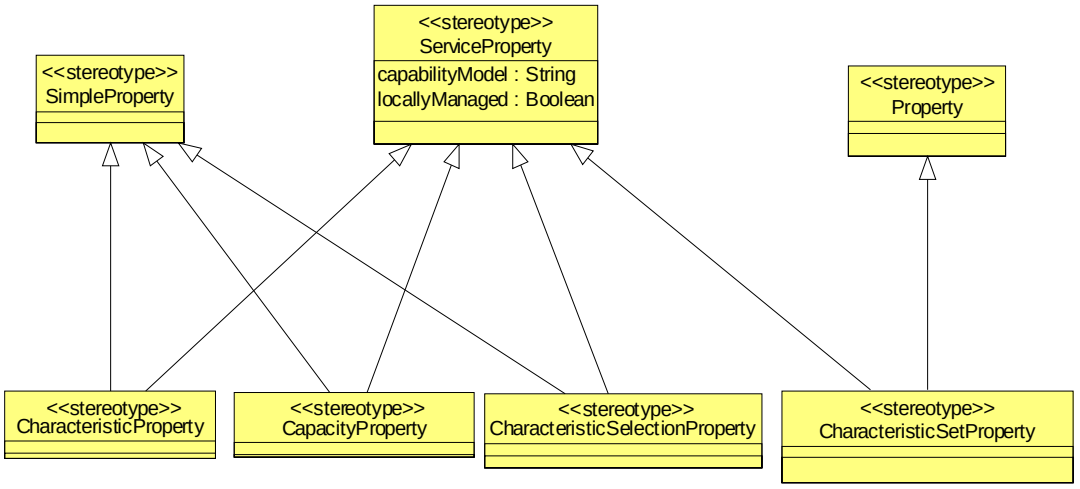

Figure 7.3 - ServiceProperty Definition

Attributes

- `capabilityModel: String`
  The capabilityModel attribute identifies the CapabilityModel to be used for this ServiceProperty.
- `locallyManaged: Boolean`
  The locallyManaged attribute indicates whether a Service manages this capability or capacity. A value of True means a Service manages the capacity otherwise it does not.

Constraints

- ServiceProperty shall have a value (not null) when the locallyManaged attribute value is false. The corresponding OCL is as follows:
  context ServiceProperty inv requiredValue: self.locallyManaged and self.value.nonEmpty()

Semantics

The ServiceProperty's capabilityModel attribute indicates the type of capabilityModel to be used to determine if the ServiceProperty can satisfy the deployment requirement.

7.1.3.13 SimpleProperty

Description

The SimpleProperty is a type of Property that defines a primitive data type (e.g., character, ULong, string, etc.).

Constraints

- The value shall comply with the property type.
- The type shall be limited to the primitive types (Boolean, Character, Float, Double, Long, LongDouble, LongLong, ObjectReference, Octet, Short, String, ULong, ULongLong, UShort, WChar, WString) and EnumerationProperty. The corresponding OCL is as follows:

  def: primTypes : Set = {Boolean, Character, Float, Double, Long, LongDouble, LongLong, ObjectReference, Octet, Short, String, ULong, ULongLong, UShort, WChar, WString}
  context SimpleProperty inv validtype: primTypes->exists(t : self.oclIsTypeOf(t))
- The range attribute min and max values shall be compatible with the type and max is greater than or equal to min. The corresponding OCL is as follows:

  context simpleProperty inv validrange: self.range.max >= self.range.min
  inv validmax: (self.oclIsTypeOf(Boolean) or self.oclIsTypeOf(Character) or self.oclIsTypeOf(ObjectReference) or self.oclIsTypeOf(String) or self.oclIsTypeOf(WString) or self.oclIsTypeOf(WChar)) or ((self.oclIsTypeOf(Float) and (self.range.max >= 0 or self.range.max <= 255)) or (self.oclIsTypeOf(Double) and (self.range.max >= 0 or self.range.max <= 255)) or (self.oclIsTypeOf (Long) and (self.range.max >= -2^{31} or self.range.max <= 2^{31} - 1)) or (self.oclIsTypeOf (LongDouble) and (self.range.max >= 0 or self.range.max <= 255)) or (self.oclIsTypeOf(LongLong) and (self.range.max >= -2^{63} or self.range.max <= 2^{63} - 1)) or (self.oclIsTypeOf(Octet) and (self.range.max >= 0 or self.range.max <= 255)) or (self.oclIsTypeOf(Short) and (self.range.max >= -2^{15} or self.range.max <= 2^{15} - 1)) or (self.oclIsTypeOf(ULong) and (self.range.max >= 0 or self.range.max <= 2^{32} - 1)) or (self.oclIsTypeOf(ULongLong) and (self.range.max >= 0 or self.range.max <= 2^{64} - 1)) or (self.oclIsTypeOf(UShort) and (self.range.max >= 0 or self.range.max <= 2^{16} - 1)))
  inv validmin: (self.oclIsTypeOf(Boolean) or self.oclIsTypeOf(Character) or self.oclIsTypeOf(ObjectReference) or self.oclIsTypeOf(String) or self.oclIsTypeOf(WString) or self.oclIsTypeOf(WChar)) or ((self.oclIsTypeOf(Float) and (self.range.min >= 0 or self.range.min <= 255)) or (self.oclIsTypeOf(Double) and (self.range.min >= 0 or self.range.min <= 255)) or (self.oclIsTypeOf (Long) and (self.range.min >= -2^{31} or self.range.min <= 2^{31} - 1)) or (self.oclIsTypeOf (LongDouble) and (self.range.min >= 0 or self.range.min <= 255)) or (self.oclIsTypeOf(LongLong) and (self.range.min >= -2^{63} or self.range.min <= 2^{63} - 1)) or (self.oclIsTypeOf(Octet) and (self.range.min >= 0 or self.range.min <= 255)) or (self.oclIsTypeOf(Short) and (self.range.min >= -2^{15} or self.range.min <= 2^{15} - 1)) or (self.oclIsTypeOf(ULong) and (self.range.min >= 0 or self.range.min <= 2^{32} - 1)) or (self.oclIsTypeOf(ULongLong) and (self.range.min >= 0 or self.range.min <= 2^{64} - 1)) or (self.oclIsTypeOf(UShort) and (self.range.min >= 0 or self.range.min <= 2^{16} - 1)))
- The value shall comply with the range when the type is a numeric. The corresponding OCL is as follows:

context SimpleProperty inv validvaluerange: (self.oclIsTypeOf(Boolean) or self.oclIsTypeOf(Character) or self.oclIsTypeOf(ObjectReference) or self.oclIsTypeOf(String) or self.oclIsTypeOf(WString) or self.oclIsTypeOf(WChar)) or ((self.oclIsTypeOf(Float) or self.oclIsTypeOf(Double) or self.oclIsTypeOf (Long) or self.oclIsTypeOf (LongDouble) or self.oclIsTypeOf(LongLong) or self.oclIsTypeOf(Octet) or self.oclIsTypeOf(Short) or self.oclIsTypeOf(ULong) or self.oclIsTypeOf(ULongLong) or self.oclIsTypeOf(UShort)) and (self.value <= self.range.max and self.value >= self.range.min))

7.1.3.14 StructProperty

Description

The StructProperty is a type that contains a list of SimpleProperties.

Constraints

- Each StructProperty's attribute name must be unique within the StructProperty. The corresponding OCL is as follows:
  context StructProperty inv: self.allAttributes()->isUnique(a | a.name)
- Each StructProperty's attribute shall be a SimpleProperty or primitive type. The corresponding OCL is as follows:
  context StructProperty inv: self.allAttributes()->forAll(a | a.stereotype.name = 'SimpleProperty' or primTypes->exists(t : a.oclIsTypeOf(t)))
- The multiplicity for each StructProperty's attribute shall be one. The corresponding OCL is as follows:
  context StructProperty inv: self.allAttributes()->forAll(a | a.size = 1)

7.1.3.15 TestDefProperty

Description

The TestDefProperty, a type of Data Type as shown in Table 7.1, provides the capability to define the input parameters for the test and the results that can be returned for a test.

Constraints

- The attribute shall be InputValueProperty or ResultValuePoperty stereotypes. The corresponding OCL is as follows:
  context TestDefProperty inv: self.all Attributes()->forAll(a | a.stereotype.name = 'InputValueProperty' or a.stereotype.name = 'ResultValueProperty')
- There shall be at least one ResultValueProperty attribute defined. The corresponding OCL is as follows:
  context TestDefProperty inv: self.allAttributes()->exists(a | a.stereotype.name = 'ResultValueProperty')
- Each Attribute name shall be unique. The corresponding OCL is as follows:
  context TestDefProperty inv: self.allAttributes()->isUnique(a | a.name)
- Each Attribute value shall be specified (not null). The corresponding OCL is as follows:
  context TestDefProperty inv: self.allAttributes()->forAll(a | a.value->notEmpty())

7.1.3.16 TestProperty

Description

The TestProperty, a type of Property as shown in Table 7.1, provides the capability to define the input parameters for the test and the results that can be returned for a test.

Constraints

- The type for a TestProperty shall be stereotype as TestDefProperty.
- The ID attribute is mandatory for a TestProperty.
- The ID attribute shall be an unsigned long string value.

7.1.4 Interface and Port Stereotypes

This section defines the port, property, and interface stereotypes for interfaces and components as depicted in Table 7.2. Port stereotypes categorize the function of the various ports within a component and the type of interfaces associated with these ports. Interface stereotypes categorize the type of interface provided by or used by components. Property stereotypes are defined for interface and component attributes to indicate the type of property visually and how the property is going to be managed. These stereotypes are used to define elements in the UML Profile for Component Framework, PIM Facilities, and by application and device component developers.

Table 7.2 - Interface & Port Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
Constant	Property	N/A	N/A	isStatic is true.	Represents a static property that is visible.
ControlPort	Port	Port		Only associated with IControl interfaces.	Represents a port for control management.
DataControlPort	Port	Port		Only associated with IDataControl interfaces.	Represents a port that sends or receives data with control using an IDataControl Interface
DataPort	Port	Port		Only associated with IData interfaces.	Represents a port that sends or receives data using an IDATA interface
Exception	Classifier				Represents an exception type that is raised by an operation and defined in an interface or package.
IControl	Interface				Provides the mechanism for sending or receiving control
IData	Interface				Provides the mechanism to send data.
IDataControl	Interface				Provides the mechanism to send data with control.
IStream	Interface				Provides the mechanism to manage streams
IStreamControl	Interface				Provides the mechanism to manage streams that contain control information in addition to user data
Oneway	Operation			The operation may not have any return parameters or exceptions.	Represents an asynchronous interaction between a caller and a receiver where the caller proceeds immediately after making the call.

Table 7.2 - Interface & Port Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
Port	Port	N/A		A port can only be associated with one interface type.	Represents a port with an interface type constraint.
ProvidesPort	Port	Port		isService is true.	Indicates a port that provides a specific type of service.
ReadOnly	Property	N/A		IsReadOnly = True	Represents a queryable property for an interface or component that has an associated get operation.
ReadWrite	Property	N/A		IsReadOnly = False	Represents a configurable and queryable property for an interface or component that has associated set and get operations.
RequiresPort	Port	Port	connection Threshold	isService is false.	Indicates a port that requires a specific type of service.
ServicePort	Port	Port			Represents a port that provides or uses a Service
StreamControl Port	Port	Port	SAP, Address	Only associated with IStreamControl interfaces.	Represents a port that sends or receives a continuous data stream, with occasional control information
StreamPort	Port	Port	SAP, Address	Only associated with IStream interfaces.	Represents a port that sends or receives continuous streaming of data

7.1.4.1 RequiresPort

Description

The RequiresPort defines a port that uses a specific service.

Attributes

- `connectionThreshold: ULong`
  The ConnectionThreshold attribute determines the number of connections allowed for a requires port.

7.1.4.2 StreamPort

Description

The StreamPort defines a streaming port that receives or sends continuous data.

Attributes

- `sap: ULong`
  The sap (Service Access Point) attribute contains a SAP identifier
- `address: OctetSequence`
  The address attribute contains address information.

Constraints

- The StreamPort shall only be associated with IStream interfaces.

7.1.5 Resource Components

This section defines the interfaces for a ResourceComponent along with component stereotypes. The Resource Component stereotypes are extensions of the UML 2.0 Component (UML2.0::Components::BasicComponents) classifiers as described in Section 7.1.5.2. Figure 7.4 depicts the base interfaces used in defining software components for applications and logical devices. The following subsection describe the details of these base interfaces (7.1.5.1), which are management interfaces for components along with the component stereotypes (7.1.5.2).

7.1.5.1 Resource Components Interfaces

This section defines the resource component modelLibrary interfaces contained in the profile definition as shown in Figure 7.4, which are: ComponentIdentifier, ControllableComponent, LifeCycle, PortConnector, PortSupplier, PropertySet, Resource, ResourceFactory, and TestableObject that are described in the following subsections. These interfaces provide basic management interfaces for components.

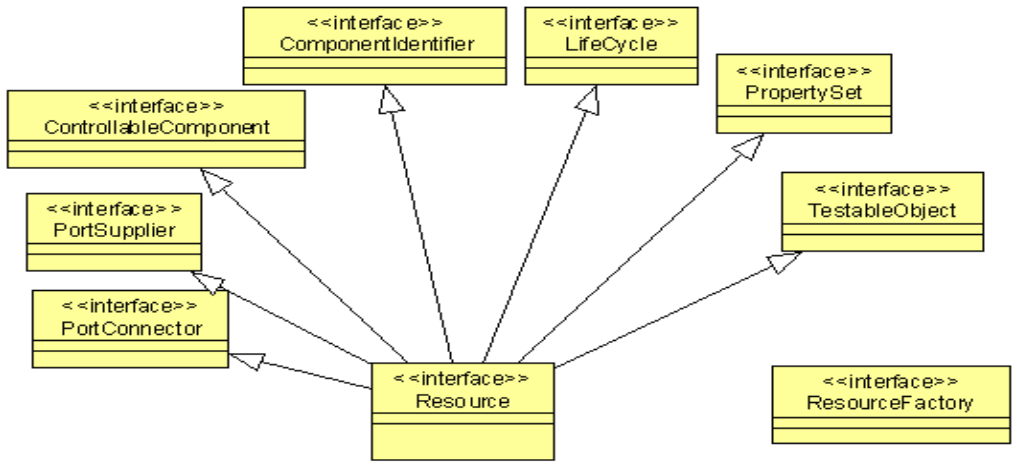

Figure 7.4 - Resource Interfaces Overview

Types and Exceptions

- `<<exception>>UnknownProperties (invalidProperties : Properties)`
  The UnknownProperties <<exception>> indicates a set of properties unknown by a component.

7.1.5.1.1 ComponentIdentifier

Description

The ComponentIdentifier interface defines the identifier operation for a component.

Attributes

- `<<readonly>>identifier: String`

The unique identifier for a component.

7.1.5.1.2 ControllableComponent

Description

The ControllableComponent interface defines the generic operations for controlling a component.

Attributes

- `<<readonly>>started: Boolean`
  The started attribute indicates if a component has been started or not. A value of True indicates the start has been performed successfully. A value of False means stop mode of operation.

Operations

- `start(): {raises = (StartError)}`
  The start operation is provided to command a component implementing this interface to start internal processing. The start operation puts the component in an operating condition. The behavior when entering into an operating condition is component implementation specific. The component implementation's current internal state (e.g., current settings of data structures, memory allocations, hardware device configurations, etc.) is used as the operational starting point. This operation does not return a value. The start operation shall raise the StartError exception if an error occurs while starting the component.
- `stop(): {raises = (StopError)}`
  The stop operation is provided to command a component implementing this interface to stop internal processing. The stop operation disables all current operations and puts a component in a non-operating condition. The behavior when exiting the operating state is component implementation specific. This operation does not return a value. The stop operation shall raise the StopError exception if an error occurs while stopping the component.

Types and Exceptions

- `<<exception>>StartError`
  The StartError, a type of SystemException, indicates that an error occurred during an attempt to start the Resource. The error number value (e.g., CF_EDOM, CF_EPERM, CF_ERANGE) and message is component-dependent, providing additional information describing the reason for the error.
- `<<exception>>StopError`
  The StopError, a type of SystemException, indicates that an error occurred during an attempt to stop the Resource. The error number (e.g., CF_ECANCELED, CF_EFAULT, CF_EINPROGRESS) and message is component-dependent, providing additional information describing the reason for the error.

7.1.5.1.3 LifeCycle

Description

The LifeCycle interface defines the generic operations for initializing or releasing instantiated component-specific data and/or processing elements.

Operations

- `initialize(): {raises = (InitializeError)}`
  The purpose of the initialize operation is to provide a mechanism to set a component to a known initial state (e.g., data

structures may be set to initial values, memory may be allocated, hardware devices may be configured to some state, etc.). The Initialization behavior is component implementation dependent. This operation does not return a value. The initialize operation shall raise the InitializeError when an initialization error occurs.

- `releaseObject(): {raises = (ReleaseError)}`
  The purpose of releaseObject is to provide a means by which a component may be removed. The releaseObject operation shall release all internal memory allocated by the component. The releaseObject operation shall remove the component from the Operating Environment (OE). This operation does not return a value. The releaseObject operation shall raise the ReleaseError when a release error occurs.

Types and Exceptions

- `<<exception>>InitializeError(errorMessage: StringSequence)`
  The InitializeError exception indicates that an error occurred during component initialization. The errorMessage attribute is component-dependent and provides additional information describing why the error occurred.
- `<<exception>>ReleaseError(errorMessage: StringSequence)`
  The ReleaseError exception indicates that an error occurred during component releaseObject. The errorMessage attribute is component-dependent and provides additional information describing why the error occurred.

7.1.5.1.4 PortConnector

Description

The PortConnector interface provides operations for managing associations between ports. The PortConnector interface is used to connect a requires port to a provides port.

Operations

- `connectPort(in requiredPortName: String, in connection: Object, in connectionId: String): {raises = (InvalidPort, OccupiedPort)}`
  The connectPort operation shall make a connection to a component's provides port identified by its input parameters. This operation does not return a value. The connectPort operation shall support all of the requires ports identified in the component's descriptor. The connectPort operation shall raise InvalidPort when connection is an invalid connection for this Port. The connectPort operation shall raise OccupiedPort when unable to accept the connections because the Port is already fully occupied.
- `disconnectPort (in requiredPortName: String, in connectionId: String): {raises = (InvalidPort)}`
  The disconnectPort operation shall break the connection to the component. The connection is identified by requiredPortName and connectionId. The disconnectPort operation shall raise InvalidPort when the requiredPortName or connectionId passed to disconnectPort is not connected or associated with the component. This operation does not return a value.

Types and Exceptions

- `<<exception>>InvalidPort ( errorCode: UShort, msg: String)`
  The InvalidPort exception indicates one of the following errors has occurred in the specification of a Port association:
  errorCode 1 means the connection (Provides Port) component is invalid or illegal object reference
  errorCode 2 means the connectionId is not known (not used by this Port)
  errorCode 3 means the Requires Port name does not exist for this component
- `<<exception>>OccupiedPort`
  The OccupiedPort exception indicates the Port is unable to accept any additional connections.

Constraints

- The PortConnector interface shall support all the uses or requires ports as specified in the component's descriptor that realizes this interface.

Semantics

A component realizes operations for transferring data and control. The component also establishes the meaning of its data and control values. Examples of how components may use ports include: push or pull, synchronous or asynchronous, mono- or bi-directional, and whether to use flow control (e.g., pause, start, stop). The nature of PortConnector, fan-in, fan-out, or one-to-one is component dependent. A requires port may support several connections. How components' ports are connected is described in a component assembly descriptor. The input connectionId is a unique identifier used by disconnectPort when breaking this specific connection from the requires port identified by the input requiredPortName. The connectionId is unique at the requires port level.

7.1.5.1.5 PortSupplier

Description

This interface provides the getProvidesPorts operation for components that have provides ports.

Operations

- `getProvidesPorts(inout ports: PortSequence): {raises = ( UnknownPorts )}`
  The getProvidesPorts operation provides a mechanism to obtain a component's provides ports in form of a sequence of name/value pairs, where each name corresponds to a provides port's name and the corresponding value is the provides port reference to be returned. The getProvidesPorts operation shall return all the component provides ports if the ports argument is zero size. The getProvidesPorts operation shall return only those provides ports specified in the ports argument if the ports argument is not zero size. The getProvidesPorts operation shall support all of the provides ports identified in the component's descriptor. The getProvidesPorts operation shall raise UnknownPorts when one or more provides port names being requested are not known by the component.

Types and Exceptions

- `PortType (name: String, providesPort: Object)`
  PortType defines a structure that associates a provides name with a provides port reference.
- `PortSequence`
  PortSequence provides an unbounded sequence of PortType.
- `<<exception>>UnknownPorts (invalidPorts: StringSequence)`
  The UnknownPorts exception is raised when one or more provides ports being requested are not known by the component. The invalidPorts attribute returned indicates the requested provides ports that were invalid.

Constraints

- The PortSupplier interface shall support all the provided interfaces as specified in the component's descriptor that realizes this interface.

7.1.5.1.6 PropertySet

Description

The PropertySet interface defines configure and query operations to access component properties/attributes.

Operations

- `configure(in configProperties:Properties):{raises=(InvalidConfiguration, PartialConfiguration)}`
  The configure operation allows id/value pair configuration properties to be assigned to components implementing this interface. The configure operation shall assign values to the component's properties as indicated in the configProperties argument. This operation does not return a value. The configure operation shall raise PartialConfiguration when some configuration properties were successfully set and some configuration properties were not successfully set. The configure operation shall raise InvalidConfiguration when a configuration error occurs that prevents any property configuration on the component.
- `query(inout configProperties: Properties): {raises = (UnknownProperties)}`
  The query operation allows a component to be queried to retrieve its properties. The query operation shall return all the component queryable properties if the input configProperties are zero size. The query operation shall return only those id/value pairs specified in the input configProperties if the configProperties are not zero size. The query operation shall raise UnknownProperties when one or more properties being requested are not known by the component.

Types and Exceptions

- `<<exception>>InvalidConfiguration (msg: String, invalidProperties: Properties)`
  The InvalidConfiguration exception indicates the configuration of a component has failed (no configuration at all was done). The msg attribute is component-dependent, providing additional information describing the reason why the error occurred. The returned invalidProperties attribute indicates the properties that were not accepted by the component.
- `<<exception>>PartialConfiguration (reasons: StringSequence,invalidProperties: Properties)`
  The PartialConfiguration exception indicates the configuration of a Component was partially successful. The reasons attribute is component-dependent, providing additional information describing the reasons why the error occurred. The returned invalidProperties attribute indicates the properties that were not accepted by the component.

Constraints

Valid properties for the configure operation shall be ConfigureProperty(s).

Valid properties for the query operation shall be:

- ConfigureProperty and QueryProperty properties, or
- ServiceProperty properties whose locallyManaged attribute value is True, or
- ExecutableProperty properties whose queryable attribute value is True.

The value attribute for each PropertyValue in the Properties shall be in its native form as specified by the Property definition.

The PropertySet interface shall support the configure and query type properties as specified in the component's descriptor that realizes this interface.

The mapping to the ConfigureProperty and QueryProperty types to the ResourceComponent's configure and query's operation properties parameter are:

1. A SimpleProperty or primitive type corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the Configure or Query Property's name or ID attribute and the PropertyValue item's value matches the Configure or Query Property's value attribute but is converted to a format that agrees with the Configure or Query Property's type attribute.

2. A SimpleProperty sequence or primitive type sequence corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the Configure or Query Property's name or ID attribute and the PropertyValue item's value matches the Configure or Query Property's values attribute that is converted to a primitive sequence type (as described in Section 7.1.1 "Base Types"), which agrees with the Configure or Query Property's type attribute.

3. A StructProperty corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the Configure or Query Property's name or ID attribute and the PropertyValue item's value contains a Properties type, where each StructProperty's SimpleProperty (id, value) corresponds to a Properties element in the list as described in item 1 above. The Properties element list size is based on the number of StructProperty's SimpleProperty items.

4. StructProperty sequence corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the Configure or Query Property's name or ID attribute and the PropertyValue item's value contains a Properties type. The Properties element list size is based on the number of StructSequenceProperty's structValues attribute. Each item in the Properties element also contains a Properties type that is used to contain a structValue (StructProperty) as described in item 3 above.

The ExecutableProperty, CapacityProperty, and CharacteristicProperty shall follow the SimpleProperty format for the query operation. The CharacteristicSelectionProperty shall follow the SimpleProperty sequence format for the query operation. The CharacteristicSetProperty shall follow the StuctProperty sequence format for the query operation.

Semantics

The type of property is known by its name or ID attribute.

7.1.5.1.7 Resource

Description

The Resource interface, as shown in Figure 7.5, provides a common API for the control and configuration of software components (applications and device). The Resource is a specialization of the ComponentIdentifier, ControllableObject, LifeCycle, PortSupplier, PortConnector, PropertySet, and TestableObject interfaces that are described in the following subsections. These interfaces provide basic management interfaces for a component.

Constraints

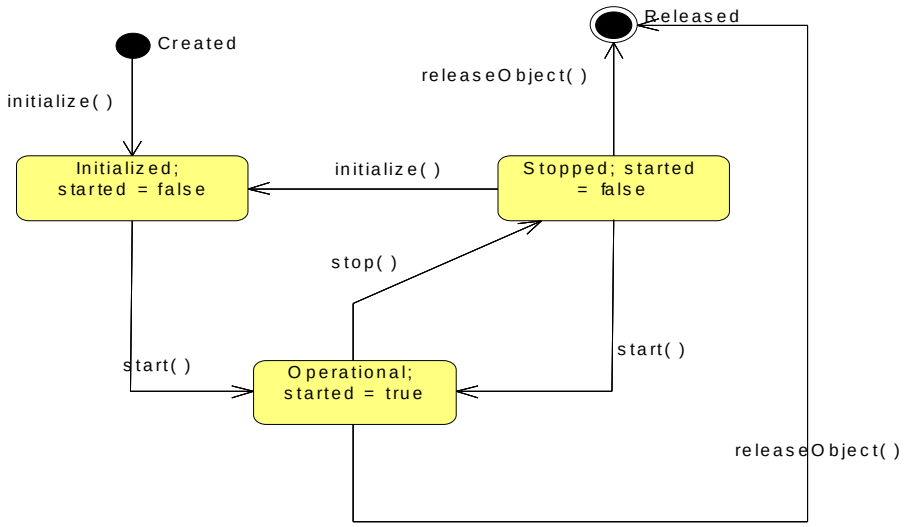

Figure 7.5 - Resources Interfaces Overview

A realization of the Resource interface shall result in a specialized clarification of the inherited ControllableObject, Lifecycle, and PortConnector interface behaviors that is consistent with the following items and Figure 7.5:

- A StartError exception shall be generated if the start operation from ControllableObject is called before at least one call is made to the initialize operation from Lifecycle.
- An InitializeError shall be generated if initialize operation from Lifecycle is called while the Resource component's ControllableObject started attribute is true.
- The behavior of the stop operation from ControllableObject shall maintain the component's current configuration to allow subsequent start operations to resume from configuration present at stop, assuming no other LifeCycle, PortConnector, PropertySet, or TestableObject operations are exercised while stopped.
- The disconnectPort operation shall perform any cleanup associated with object being disconnected before completing the disconnect operation. The specific cleanup processing is Resource component implementation dependent.
- Use of the releaseObject operation from LifeCycle while the Resource component's ControllableObject started attribute is true shall result in a behavior consistent with first initiating the Resource's stop operation, then calls to disconnectPort on each of the Resource component's ports, and then the releaseObject behavior itself.

Semantics

The Resource PropertySet implementation is not inhibited by the stop operation. However, the configure behavior impacts to the Resource while stopped are limited in scope to updating internal data structures, in preparation for the next start operation.

The Resource Port Supplier and Port Connector implementation is not inhibited nor impacted by the stop operation, all methods of these two interfaces are supported, behavior unchanged. However, the behavior of the provided ports themselves is impacted. The impact of the stop is port implementation specific. The usual case is that port behavior is halted upon being stopped (started attribute is false). Resource component usage of connected port capabilities will cease and any access to provided port capabilities would result in error notification. An example would be an IO port that, upon

being stopped, prevents further data from being pushed out and allows no further data to be pushed in. A notable exception would include status providing ports that would remain active even while stopped to maintain good standing with observing components.

7.1.5.1.8 ResourceFactory

Description

The ResourceFactory interface provides an optional mechanism for the management of ResourceComponents.

Operations

- `createResource(in resourceId: String, in qualifiers: Properties, Return ResourceComponent): {raises = (CreateResourceFailure)}`
  The createResource operation provides the capability to create a ResourceComponent or retrieve an existing ResourceComponent. The resourceId parameter is the identifier for ResourceComponent. The qualifiers are parameter values used by the ResourceFactory in creation of the ResourceComponent. The qualifiers may be used to identify, for example, specific subtypes of a ResourceComponent created by a ResourceFactory.

  If no ResourceComponent exists for the given resourceId, the createResource operation shall create a ResourceComponent. otherwise the createResource operation shall return an existing ResourceComponent whose identifier attribute matches the input resourceId.The createResource operation shall assign the given resourceId to a new ResourceComponent's identifier attribute.

  The createResource operation shall raise the CreateResourceFailure exception when it cannot create the ResourceComponent and cannot find an existing ResourceComponent that contains the resourceId.

- `releaseResource(in resourceId: String): {raises = (InvalidResourceId)}`
  The releaseResource operation provides the mechanism of releasing the Resource in the environment on the server side. The releaseResource operation shall release the ResourceComponent from the ResourceFactory as identified by the input resourceId parameter when the number of create requests matches the number of release requests for the input resourceId.

  The releaseResource operation shall raise the InvalidResourceId exception if an invalid resourceId is received.

- `Shutdown(): {raises = (ShutdownFailure)}`
  The shutdown operation provides the mechanism for releasing the ResourceFactoryComponent from the environment on the server side. The shutdown operation results in the ResourceFactoryComponent being unavailable to any subsequent calls to its component reference (i.e., it is released from the environment). The shutdown operation shall raise the ShutdownFailure exception if unable to terminate the ResourceFactoryComponent.

Types and Exceptions

- `<<exception>>CreateResourceFailure`
  The CreateResourceFailure exception, a type of System Exception, indicates that the createResource operation failed to create the Resource. The error number indicates an ErrorNumberType value (e.g., CF_NOTSET, CF_EBADMSG, CF_EINVAL, CF_EMSGSIZE, CF_ENOMEM). The message is component-dependent, providing additional information describing the reason for the error.

- `<<exception>>InvalidResourceId`
  The InvalidResourceId exception indicates the resourceId does not exist in the ResourceFactoryComponent.

- `<<exception>>ShutdownFailure (msg: String)`
  The ShutdownFailure exception indicates that the shutdown method failed to release the ResourceFactoryComponent from the operating environment due to the fact the Factory still contains ResourceComponents. The message is component-dependent, providing additional information describing why the shutdown failed.

Constraints

The created ResourceComponent's identifier attribute shall be the resourceId parameter value.

Semantics

A ResourceFactory interface is used to create and release a ResourceComponent.

7.1.5.1.9 TestableObject

Description

The TestableObject interface defines a set of operations that can be used to test component implementations.

Operations

- `runTest(in testId:ULong, inout testValues:Properties):{raises=(UnknownTest, UnknownProperties)}`
  The runTest operation allows components to be "blackbox" tested. This allows Built-In Test (BIT) to be implemented and this provides a means to isolate faults (both software and hardware) within the system. The runTest operation shall use the testId parameter to determine which of its predefined test implementations should be performed. The testValues parameter Properties (id/value pair(s)) shall be used to provide additional information to the implementation-specific test to be run. The runTest operation shall return the result(s) of the test in the testValues parameter.

  The runTest operation shall raise UnknownTest when there is no underlying test implementation associated with the input testId given.

  The runTest operation shall raise UnknownProperties when the input parameter testValues contains input test parameters that are invalid. The UnknownProperties' invalidProperties attribute contains the invalid inputValues properties id(s) that are not known by the component or the value(s) that are out of range.

Types and Exceptions

- `<<exception>>UnknownTest`
  The UnknownTest exception indicates the requested testId for a test to be performed is not known by the component.

Constraints

The TestableObject interface shall support all the TestProperty(s) as stated in the component's descriptor that realizes this interface.

The format for a TestProperty's InputValueProperty(s) and ResultValueProperty(s) for a test shall be as described for a SimpleProperty in the PropertySet (Section 7.1.5.1.6).

Semantics

The testid parameter corresponds to the name or ID attribute of the TestProperty. Each TestProperty's InputValueProperty maps to the testValues parameter for input to the test. Each TestProperty's ResultValueProperty maps to the testValues, which are the return values for the test being performed.

7.1.5.2 Resource Components Stereotypes

This section defines the resource component stereotypes contained in the profile definition as shown in Table 7.3, which are: ResourceComponent, ResourceFactoryComponent, and Component that are described in the following subsections. These stereotypes provide the basic component definitions for component developers.

Table 7.3 - Resource Components Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
Component	Component	N/A		See constraints in section below	Represents the base definition for all Components.
ResourceComponent	Component	Component		See constraints in section below	Represents a component of an application.
ResourceFactoryComponent	Component	Component		See constraints in section below	Manages ResourceComponent(s)

7.1.5.2.1 Component

Description

The Component, as shown in Table 7.3, is extension of the UML component. Figure 7.6 depicts the relationships for any software component.

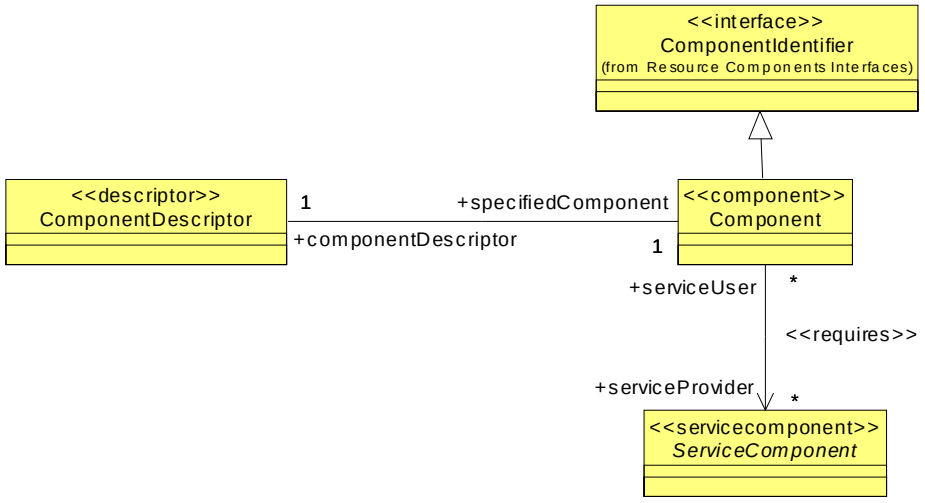

Figure 7.6 - Component M1 Illustration

M1 Associations

- `componentDescriptor: ComponentDescriptor [1]`
  A Component has at least one descriptor that describes the component's characteristics such as ports and properties.
- `serviceProvider: ComponentService [*]`
  A Component can be optionally associated with zero to many Services.

Constraints

A Component shall realize the ComponentIdentifier interface.

If a component has test properties, then the component shall realize the TestableObject Interface.

If a component has configure and/or query properties, then the component shall realize the PropertySet interface.

If a component has provides ports, then the component shall realize the PortSupplier interface.

If a component has uses ports, then the component shall realize the PortConnector interface.

Semantics

Component shall implement a ConfigureProperty with a name of PRODUCER_LOG_LEVEL when connected to a log service. The PRODUCER_LOG_LEVEL ConfigureProperty provides the ability to "filter" the log message output of a Component. This property may be configured via the PropertySet interface to output only specific log levels.

Components shall output only those log records to a LogService that correspond to enabled log level values in the PRODUCER_LOG_LEVEL attribute. Log levels that are not in the PRODUCER_LOG_LEVEL attribute are disabled. Components shall use their identifier attribute in the log record output to the LogService. Components shall operate normally in the case where the connections to a LogService are nil or an invalid reference.

A Component could also take on additional behavior for life cycle management by realizing the LifeCycle interface that would be used during deployment and teardown of a component. A Component may also realize the ControllableComponent interface to provide overall management control of the component.

7.1.5.2.2 ResourceComponent

Description

The ResourceComponent, as shown in Table 7.3, provides the component definition for software resource component. Figure 7.7 depicts the property associations for a ResourceComponent.

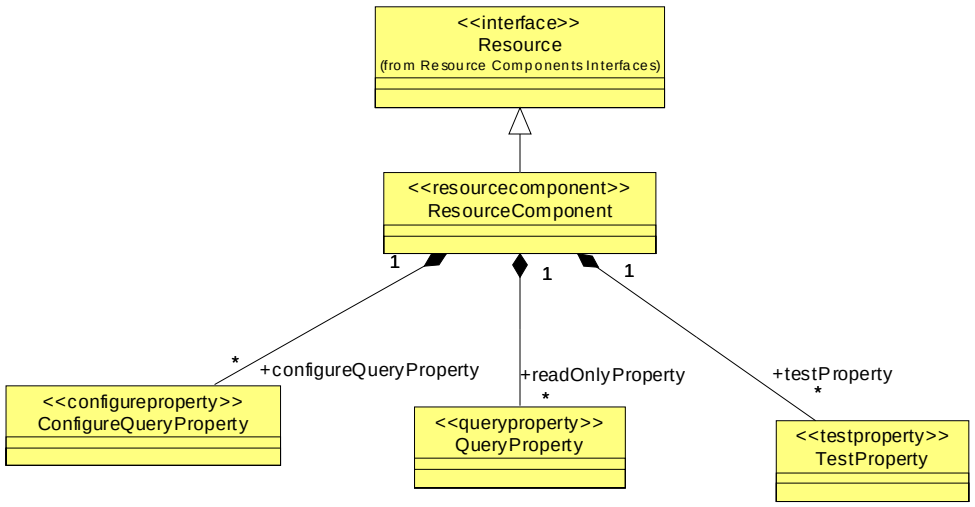

Figure 7.7 - ResourceComponent M1 Illustration

M1 Associations

- `configureQueryProperty: ConfigureProperty [*]`
  A ResourceComponent may have zero to many configurable and queryable properties.
- `readOnlyProperty: QueryProperty [*]`
  A ResourceComponent may have zero to many query properties.
- `testProperty: TestProperty [*]`
  A ResourceComponent may have zero to many test properties.

Constraints

- The ResourceComponent shall realize the Resource interface.

Semantics

The ResourceComponent configure, query, and start operations are not inhibited by the stop operation.

7.1.5.2.3 Resource Factory Component

Description

The ResourceFactory class, as shown in Table 7.3, provides an optional mechanism for the management of Resources. Figure 7.8 depicts the associations for a ResourceFactoryComponent, which are described below.

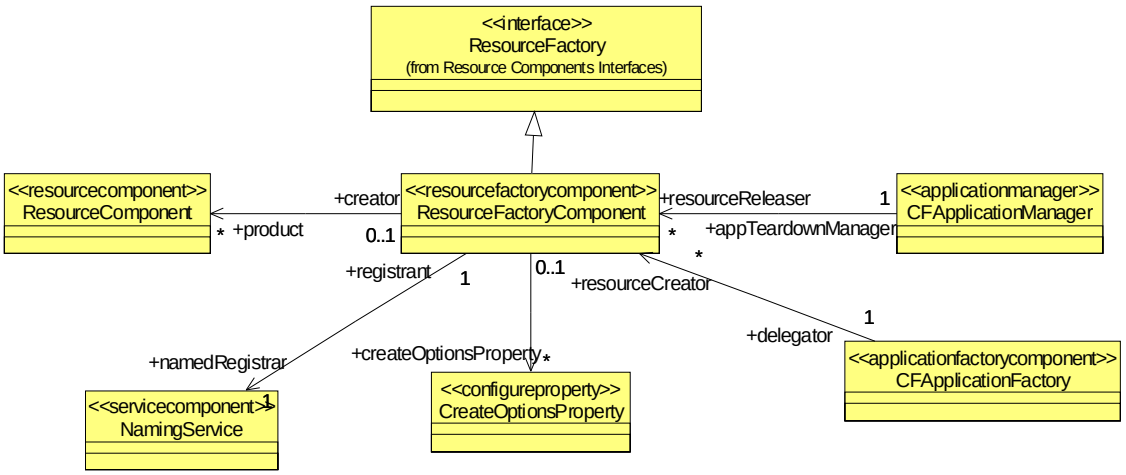

Figure 7.8 - ResourceFactoryComponent M1 Illustration

M1 Associations

- `createOptionsProperty: ConfigureProperty [*]`
  These are the properties that a ResourceFactoryComponent understands and can be used when creating up a Resource.
- `namedRegistrar: NamingService`
  The NamingService that contains a named ResourceFactoryComponent reference.
- `product: Resource [*]`
  A ResourceComponent can be created from a ResourceFactoryComponent.

Constraints

The ResourceFactoryComponent shall realize the ResourceFactory interface.

Semantics

A ResourceFactoryComponent is used to create and release a ResourceComponent. When multiple clients have obtained a reference to the same ResourceComponent, the ResourceFactoryComponent must not release the ResourceComponent until release requests have been received from all the clients that issued the create request. Application and Waveform developers are not required to use ResourceFactoryComponents for their application definition. ResourceFactoryComponent provides the mechanism of creating separate process threads for each component created in the ResourceFactory.

7.1.6 Device Components

The Device Components sections define the set of interfaces and component stereotypes used to communicate and manage physical devices. The component stereotypes are depicted in Table 7.4, which are extensions of the UML Component. The following subsections describe the details of logical device interfaces (7.1.6.1) and component stereotypes (7.1.6.2).

7.1.6.1 Device Component Interfaces

This section defines the modelLibrary device component interfaces contained in the profile definition as shown in Figure 7.9, which are: DeviceComposition, Device, LoadableDevice, and ExecutableDevice that are described in the following subsections. These interfaces provide basic management interfaces for physical devices.

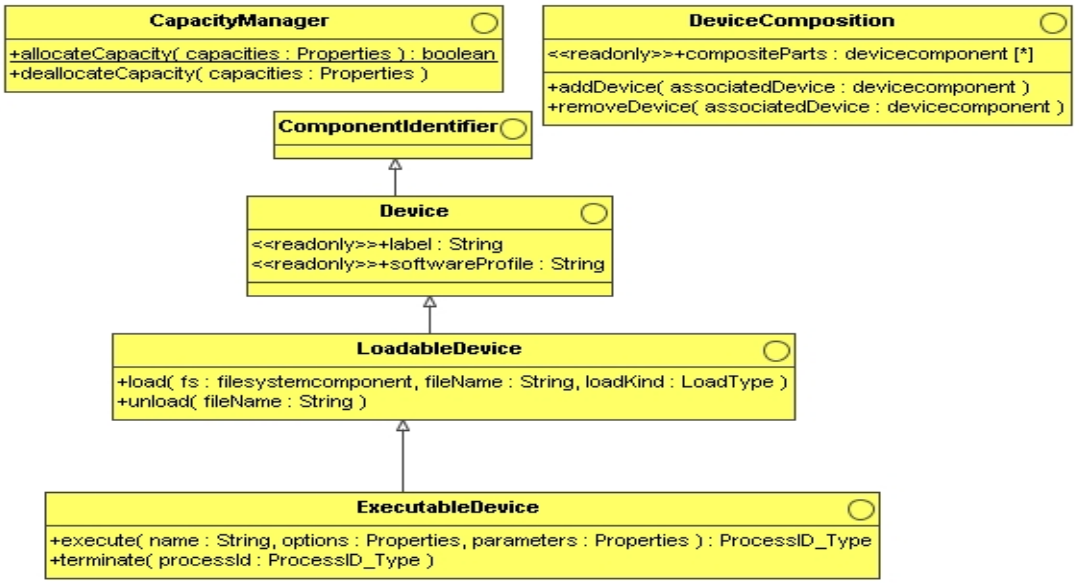

Figure 7.9 - Device Component Interfaces Definition

Types and Exceptions

- `<<exception>>InvalidState (msg:String)`
  The InvalidState exception indicates that the device is not capable of the operation being attempted due to its state(s), (e.g., admin, operational, or usage).

7.1.6.1.1 CapacityManager

Description

The CapacityManager, as shown in Figure 7.9, defines an interface that provides the mechanisms for managing capacities.

Operations

- `allocateCapacity (in capacities: Properties, return Boolean): {raises = ( InvalidCapacity, InvalidState )}`
  The allocateCapacity operation provides the mechanism to request and allocate capacity from the DeviceComponent. The allocateCapacity operation shall reduce the current capacities of the DeviceComponent based upon the input capacities parameter. The allocateCapacity operation is valid when the adminState attribute is UNLOCKED, operationalState attribute is ENABLED, and usageState attribute is not BUSY.

  The allocateCapacity operation shall set the usageState attribute to BUSY, when the DeviceComponent determines that it is not possible to allocate any further capacity. The allocateCapacity operation shall set the usageState attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation.

The allocateCapacity operation shall return “True” if the capacities have been allocated, or “False” if not allocated.

The allocateCapacity operation shall raise the InvalidCapacity exception when the capacities are invalid or the capacity values are the wrong type or ID.

The allocateCapacity operation shall raise the InvalidState exception if the Device’s adminState is not UNLOCKED or operationalState is DISABLED when invoked.

- `deallocateCapacity (in capacities: Properties): {raises = ( InvalidCapacity, InvalidState )}`
  The deallocateCapacity operation provides the mechanism to return capacities back to the Device, making them available to other users.

  The deallocateCapacity operation shall adjust the current capacities of the Device based upon the input capacities parameter. The deallocateCapacity operation is valid when the adminState is UNLOCKED or SHUTTING_DOWN and operationalState is ENABLED and usageState is not IDLE.

  The deallocateCapacity operation shall set the usageState attribute to ACTIVE when, after adjusting capacities, any of the Device’s capacities are still being used.

  The deallocateCapacity operation shall set the usageState attribute to IDLE when, after adjusting capacities, none of the Device’s capacities are being used.

  The deallocateCapacity operation does not return any value.

  The deallocateCapacity operation shall raise the InvalidCapacity exception when the capacity ID is invalid or the capacity value is the wrong type. The InvalidCapacity exception states the reason for the exception.

  The deallocateCapacity operation shall raise the InvalidState exception if the Devices adminState is LOCKED, or operationalState is DISABLED, or usageState is IDLE when invoked.

Types and Exceptions

- `<<exception>>InvalidCapacity (msg: String, capacities: Properties)`
  The InvalidCapacity exception indicates the capacities that are not valid for this device.

7.1.6.1.2 Device

Description

The Device, as shown in Figure 7.9, defines an interface that abstracts the underlying hardware. A Device (e.g., logical device) interface is a functional abstraction for a set (e.g., zero or more) of hardware devices and provides the following attributes and operations:

Attributes

- `<<readonly>>label: String`
  The readonly label attribute contains the Device’s label. The label attribute is the meaningful name given to a Device. The attribute could convey location information within the system (e.g., audio1, serial1, etc.).
- `<<readonly>>softwareProfile: String`
  The profile descriptor (data/command uses and provides ports, configure and query properties, capacity properties, status properties, etc.).

7.1.6.1.3 DeviceComposition

Description

The DeviceComposition is an interface that provides the capability to construct a composite device definition.

Attributes

- `<<readonly>>compositeParts: DeviceComponent [*]`
  The readonly compositeParts attribute shall contain a list of DeviceComponents that have been added to DeviceComposition or a zero length sequence if the composition is empty (no devices have been added or all have been removed).

Operations

- `addDevice (in associatedDevice: DeviceComponent): {raises = ( InvalidObjectReference )}`
  The addDevice operation provides the mechanism to associate a Device with a DeviceComposition. The addDevice operation shall add the input associatedDevice parameter to the compositeParts attribute when the associatedDevice does not already exist in the compositeParts attribute. The associatedDevice is ignored when duplicated. This operation does not return any value. The addDevice operation shall raise the InvalidObjectReference when the input associatedDevice is a nil DeviceComponent reference.
- `removeDevice (in associatedDevice: DeviceComponent): {raises = ( InvalidObjectReference )}`
  The removeDevice operation provides the mechanism to disassociate a DeviceComponent from a DeviceComposition. The removeDevice operation shall remove the input associatedDevice parameter from the compositeParts attribute. This operation does not return any value. The removeDevice operation shall raise the InvalidObjectReference when the input associatedDevice is a nil DeviceComponent reference or does not exist in the compositeParts attribute.

Semantics

The DeviceComposition interface provides composite behavior that can be used to add and remove DeviceComponents from a composite relationship. Composite part DeviceComponents use this interface to introduce or remove an association between themselves and a DeviceComponent that manages the composition. When the aggregating DeviceComponent that manages the DeviceComposition changes state or is being released by the releaseObject operation, its associated DeviceComponents are affected. (all DeviceComponents added to the DeviceComposition)

7.1.6.1.4 ExecutableDevice

Description

The ExecutableDevice interface, as shown in Figure 7.9, extends the LoadableDevice interface by adding execute and terminate behavior.

Attributes

- `<<constant>>PRIORITY_ID : String = "PRIORITY"`
  The PRIORITY_ID is the identifier for the Executable’s execute options parameters. The value for a priority option parameter shall be an unsigned long.

- `<<constant>>STACK_SIZE_ID: String = "STACK_SIZE"`
  The STACK_SIZE_ID is the identifier for the Executable's execute options parameter. The value for a stack size option parameter shall be an unsigned long.

- `<<constant>>CREATE_THREAD_REQUEST: String = "CREATE_THREAD"`
  The CREATE_THREAD_REQUEST is the identifier for the Executable's execute options parameter. The value for create thread request option shall be an unsigned long that indicates the thread ID to be collocated with. A zero valid indicates no thread ID collocation is indicated. A non-zero indicates the thread ID to be collocated with.

- `<<constant>>RUNTIME_REQUEST: String = "RUNTIME_REQUEST"`
  The RUNTIME_REQUEST is the identifier for the Executable's execute options parameter. The value for runtime request option shall be a string of the runtime name to be executed.

- `<<constant>>RUNTIME_OPTIONS: String = "RUNTIME_OPTIONS"`
  The RUNTIME_OPTIONS is the identifier for the Executable's execute options parameter. The value for runtime options option shall be a Base-Types::Properties. Each ID/value pair in the Properties represents a runtime option. The id indicates the option name and the value is the option value.

Operations

- `execute(in name: String, in options: Properties, in parameters: Properties, return ProcessID_Type): {raises = (InvalidState, InvalidFunction, InvalidParameters, InvalidOptions, InvalidFileName, ExecuteFail)}`
  The execute operation provides the mechanism for starting up and executing a software process or thread. A process or thread can be used to execute a runtime environment, function, or file.

- `terminate(in processId: ProcessID_Type): {raises = ( InvalidProcess, InvalidState )}`
  The terminate operation provides the mechanism for terminating the execution of a process or thread on a specific device that was started up with the execute operation. The terminate operation shall terminate the execution of the process or thread designated by the processId input parameter on the Device.

  When the last thread is terminated from a process, the terminate operation should terminate the process. The terminate operation shall raise the InvalidState exception if the Device's adminState is LOCKED or operationalState is DISABLED upon invocation.

  The terminate operation shall raise the InvalidProcess exception when the processId is not executing on the Device.

Types and Exceptions

- `<<exception>>ExecuteFail`
  The ExecuteFail exception, a type of SystemException, indicates that the Execute operation failed due to device dependent reasons. The ExecuteFail exception indicates that an error occurred during an attempt to invoke the execute function on the device. The error number indicates an ErrorNumberType value (e.g., CF_EACCES, CF_EBADF, CF_EINVAL, CF_EIO, CF_EMFILE, CF_ENAMETOOLONG, CF_ENOENT, CF_ENOMEM, CF_ENOTDIR). The message is component-dependent, providing additional information describing the reason for the error.

- `<<exception>>InvalidFunction`
  The InvalidFunction exception indicates that a function, as identified by the input name parameter, hasn't been loaded on this device.

- `<<exception>>InvalidProcess`
  The InvalidProcess exception, a type of SystemException, indicates that a process, as identified by the processID parameter, is not executing on this device. The error number indicates an ErrorNumberType value (e.g., CF_ESRCH, CF_EPERM, CF_EINVAL). The message is component-dependent, providing additional information describing the reason for the error.

- `<<exception>>InvalidParameters ( invalidParms: Properties )`
  The InvalidParameters exception indicates the input parameters are invalid on the execute operation. The InvalidParameters exception is raised when there are invalid execute parameters. Each parameter's ID and value must be a valid string type. The invalidParms attribute is a list of invalid parameters specified in the execute operation.

- `<<exception>>InvalidOptions ( invalidOpts: Properties)`
  The InvalidOptions exception indicates the input options are invalid on the execute operation. The invalidOpts attribute is a list of invalid options specified in the execute operation.

- `ProcessID_Type`
  This type, a specialization of Long, defines a process number within the system. Process number is unique to the Processor operating system that created the process.

Semantics

The execute operation shall execute the function or file identified by the input name parameter using the input parameters and options parameters when no runtime or thread options are specified. Whether the input name parameter is a function or a file name is implementation-specific.

The execute operation shall pass the input parameters (ID/value string pairs) as arguments (array of strings) to the operating system "execute/thread" function, where argument (0) is the function name, argument (1) maps to input parameters (0) id and argument (2) maps to input parameters (0) value and so forth.

The execute operation shall create a thread when the options parameter is CREATE_THREAD_REQUEST is specified. The execute operation shall create thread in the same process as the thread ID identified by the CREATE_THREAD_REQUEST value when the CREATE_THREAD_REQUEST value is not zero.

The execute operation shall create a runtime process/thread when the RUNTIME_REQUEST options is specified in the options parameter. The execute operation creates the runtime process/thread using the RUNTIME_REQUEST value. The execute operation shall pass the RUNTIME_OPTIONS as specified in the input options parameter, the input name, and arguments in the form that is compliant with runtime parameters syntax.

The execute operation shall use STACK_SIZE_ID and PRIORITY_ID options, when specified, to set the process/thread stack size and priority for the target executable. The execute operation returns a unique processID for the process or thread that it created.

The execute operation shall raise the InvalidState exception, if the Device's adminState is not UNLOCKED or operationalState is DISABLED when invoked. The execute operation shall raise the InvalidFunction exception when the function indicated by the input name parameter does not exist for the Device (e.g., not loaded on device). The execute operation shall raise the InvalidFileName exception when the file name indicated by the input name parameter does not exist for the Device (e.g., not loaded on device). The execute operation shall raise the InvalidParameters exception when the input parameters parameter item ID or value are not string types. The execute operation shall raise the InvalidOptions exception when the input options parameter does not comply with STACK_SIZE_ID, PRIORITY_ID, THREAD_CREATE_REQUEST, RUNTIME_CREATE_REQUEST, and RUNTIME_OPTIONS (described in Types section below). The execute operation shall raise the ExecuteFail exception when the operating system "execute" function for the device is not successful.

7.1.6.1.5 LoadableDevice

Description

The LoadableDevice interface, as shown in Figure 7.9, extends the Device interface by adding software loading and unloading behavior.

Operations

- `load(in fs: FileSystem, in filename: String , in loadKind: LoadType): {raises = (InvalidState, InvalidLoadKind, InvalidFileName, LoadFail )}`
  The load operation provides the mechanism for loading software on a specific device.

  The load operation shall load a file on the specified device based upon the input loadKind and fileName parameters using the input FileSystem parameter to retrieve the file.

  Multiple loads of the same input fileName do not result in an exception or a duplicate load, however the load operation should account for this attempt so that the unload operation behavior can be performed. The load operation shall raise the InvalidState exception if the Device's adminState is not UNLOCKED or operationalState is DISABLED upon invocation.

  The load operation shall raise the InvalidLoadKind exception if the input loadKind parameter is not supported.
  The load operation shall raise the InvalidFileName exception if the file designated by the input filename parameter cannot be found.

  The load operation shall raise the LoadFail exception if an attempt to load the device is unsuccessful.

- `unload(in filename: String): {raises = ( InvalidState, InvalidFileName )}`
  The unload operation provides the mechanism to unload software that is currently loaded. The unload operation shall unload the application software on the device based on the input fileName parameter. The unload operation shall perform the unload when the number of unload requests matches the number of load requests for the input filename.

  The unload operation shall raise the InvalidState exception if the adminState attribute is LOCKED or its operationalState attribute is DISABLED upon invocation.

  The unload operation shall raise the InvalidFileName exception if the file designated by the input filename parameter cannot be found.

Types and Exceptions

- `<<enumeration>>LoadType ( KERNEL_MODULE, DRIVER, DLL, EXECUTABLE,SHARED_LIBRARY)`
  The LoadType defines the type of load to be performed which are:
  - KERNEL_MODULE
  - DRIVER
  - DLL
  - EXECUTABLE,
  - SHARED_LIBRARY

- `<<exception>>InvalidLoadKind`
  The InvalidLoadKind exception indicates that the LoadableDevice is unable to load the type of file designated by the loadKind parameter.
- `<<exception>>LoadFail ( errorNumber : ErrorNumberType, msg : String)`
  The LoadFail exception indicates that the Load operation failed due to device dependent reasons. The LoadFail exception indicates that an error occurred during an attempt to load the device. The error number indicates an ErrorNumberType value (e.g., EACCES, CF_EAGAIN, CF_EBADF, CF_EINVAL, CF_EMFILE, CF_ENAMETOOLONG, CF_ENOENT, CF_ENOMEM, CF_ENOSPC, CF_ENOTDIR). The message is component-dependent, providing additional information describing the reason for the error.

7.1.6.2 Device Component Stereotypes

The component stereotypes are depicted in Table 7.4, which are extensions of the UML Component. The following subsections describe the details of logical device interfaces and component stereotypes.

Table 7.4 - Device Components Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
DeviceComponent	Component	Component, ServiceComponent		See constraints in section below	Represents a logical device that abstracts the underlying hardware.
DeviceComposition Component	Component	N/A		See constraints in section below	Provides the capability to construct composite devices.
DeviceDriver	Component	N/A			Represents a device driver that interfaces with the hardware.
ExecutableDevice Component	Component	LoadableDeviceComponent		See constraints in section below	Manages execution and termination of OS processes on a device.
LoadableDevice Component	Component	DeviceComponent		See constraints in section below	Represents a loadable device that manages loading behavior on a device.

7.1.6.2.1 DeviceComponent

Description

The DeviceComponent, as shown in Figure 7.10, defines a component that abstracts the underlying hardware. The DeviceComponent is a type of Component and ServiceComponent. A DeviceComponent (e.g., logical device) is a functional abstraction for a set (e.g., zero or more) of hardware devices.

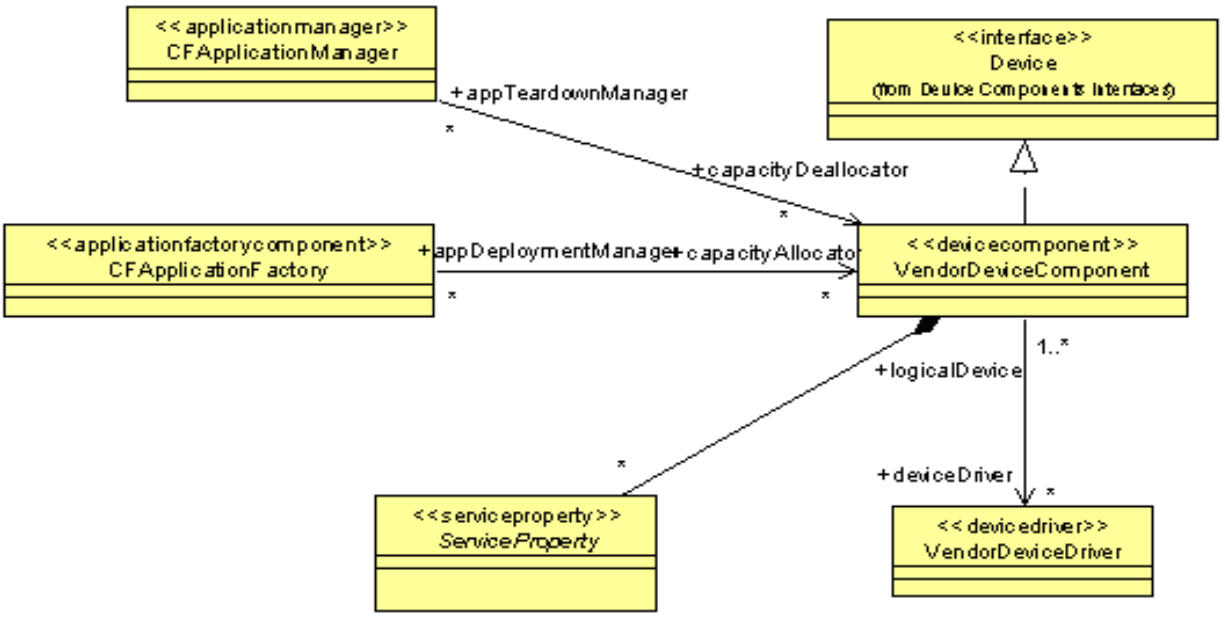

Figure 7.10 - DeviceComponent M1 Illustration

M1 Associations

- `deviceDriver: DeviceDriver [*]`
  The device drivers used by the logical device for communicating with the hardware.

Constraints

The DeviceComponent shall support the ServiceProperty capabilities and capacities properties as stated in the Device's component descriptor as specified by the softwareProfile attribute.

If the DeviceComponent has managed service properties (ServiceProperty(s) whose locallyManged attribute value is True), then the component shall realize the PropertySet interface.

If the DeviceComponent manages service properties (ServiceProperty(s) whose locallyManged attribute value is True), then the component shall realize the CapacityManager interface.

The DeviceComponent shall realize the Device interface.

Semantics

The managed capacity properties are managed by the logical device through its capacity operations and reflected by its state attributes.

The DeviceComponent contains CapacityModel(s) when the DeviceComponent contains ServiceProperty(s) whose locallyManaged attribute value is True.

The BasicDeploymentRequirement corresponds to a PropertyValue item in the allocateCapacity or deallocateCapacity capacities parameter. The PropertyValue item's id matches the BasicDeploymentRequirement's identification and the PropertyValue item's value matches the BasicDeploymentRequirement's value attribute but is converted into a format that agrees with the ServiceProperty's type attribute.

The DeploymentRequirementQualifier (Infrastructure:: Deployment:: Artifacts) corresponds to a PropertyValue item in the allocateCapacity or deallocateCapacity capacities parameter. The PropertyValue item's id matches the DeploymentRequirementQualifier's identification and the PropertyValue item's value contains a Properties type, where each DeploymentRequirementQualifier's qualifier (id, value) corresponds to a Properties element in the list. The Properties element list size is based on the number of DeploymentRequirementQualifier's qualifier items. DeploymentRequirementQualifier's qualifier value attribute is converted into a format that agrees with the CharacteristicQuery value attribute.

A DeviceComponent can be extended with additional behavior such as the Resource interfaces. A DeviceComponent that manages IO communication equipment such as serial and audio would typically be extended by realizing the Resource interface. For DeviceComponents that are composite of DeviceComponents, the composite DeviceComponent would realize the DeviceComposition interface. For DeviceComponents that are managed they would also be stereotype as ManagedServiceComponent that realizes the StateManagement interface. Typically components who manage their capacities are managed components.

If the DeviceComponent realizes the StateManagement interface and the LifeCycle interface, then setAdminState operation shall become disabled when the releaseObject operation is invoked.

The following behavior is in addition to the LifeCycle releaseObject operation behavior when realized by a DeviceComponent.

If a DeviceComponent realizes the DeviceComposition interface, then the releaseObject operation shall call the releaseObject operation on all of the DeviceComponents managed by the DeviceComponent (i.e., those DeviceComponents that are contained within the DeviceComponent's compositeParts attribute).

The releaseObject operation shall transition the DeviceComponent's adminState to SHUTTING_DOWN state when the DeviceComponent's adminState is UNLOCKED, and usageState is not IDLE or the DeviceComponent's compositeParts attribute is not empty of devices.

The releaseObject operation shall transition the DeviceComponent's adminState to LOCKED when the DeviceComponent's adminState is SHUTTING_DOWN and usageState attribute is IDLE and the DeviceComponent's compositeParts attribute is empty of devices; all composite parts have been removed.

The releaseObject operation shall transition the DeviceComponent's adminState to LOCKED when the DeviceComponent's adminState is UNLOCKED, and the usageState is IDLE and the DeviceComponent's compositeParts attribute is empty of devices; all composite parts have been removed.

The releaseObject operation shall release the DeviceComponent, when the Device's adminState transitions to LOCKED, ensuring that its usageState is IDLE and any composite parts have been removed.

If the DeviceComponent is a composite part or child of another DeviceComponent then the releaseObject operation shall cause the DeviceComponent to remove itself from the composition DeviceComponent (using the DeviceComposition reference provided as an execute property at the construction of the DeviceComponent).

If the DeviceComponent is registered with a DeviceManager, then the releaseObject operation shall unregister the DeviceComponent from its DeviceManager.

7.1.6.2.2 DeviceCompositionComponent

Description

The DeviceCompositionComponent is a component that provides the capability to construct a composite device definition.

Constraints

The DeviceCompositionComponent shall realize the DeviceComposition interface.

Semantics

The DeviceCompositionComponent component provides composite behavior that can be used to add and remove DeviceComponents from a composite relationship. Composite part DeviceComponents are provided with and use a reference to a DeviceCompositionComponent (as an instance of a DeviceComposition interface realization) to introduce or remove an association between themselves and a DeviceComponent that manages the composition. When the DeviceComponent that manages the DeviceComposition changes state or is being released by the releaseObject operation, its associated DeviceComponents are affected (all DeviceComponents added to the DeviceComposition).

7.1.6.2.3 DeviceDriver

Description

The DeviceDriver, as shown in Figure 7.11, represents a component that interfaces with the communication equipment.

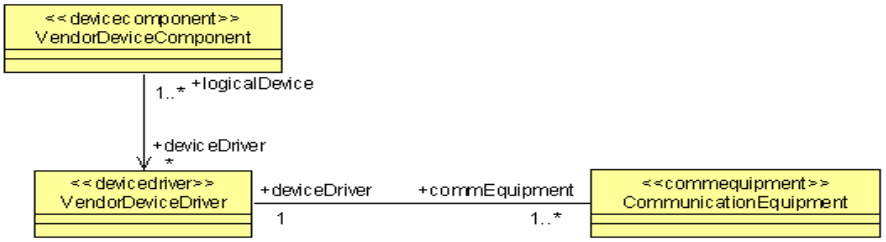

Figure 7.11 - DeviceDriver M1 Illustration

M1 Associations

- `CommEquipment: CommEquipment [1..*]`
  The device element the device driver is managing and controlling.

7.1.6.2.4 ExecutableDeviceComponent

Description

The ExecutableDeviceComponent, as shown in Figure 7.12, extends the LoadableDeviceComponent by adding execute and terminate process/thread behavior.

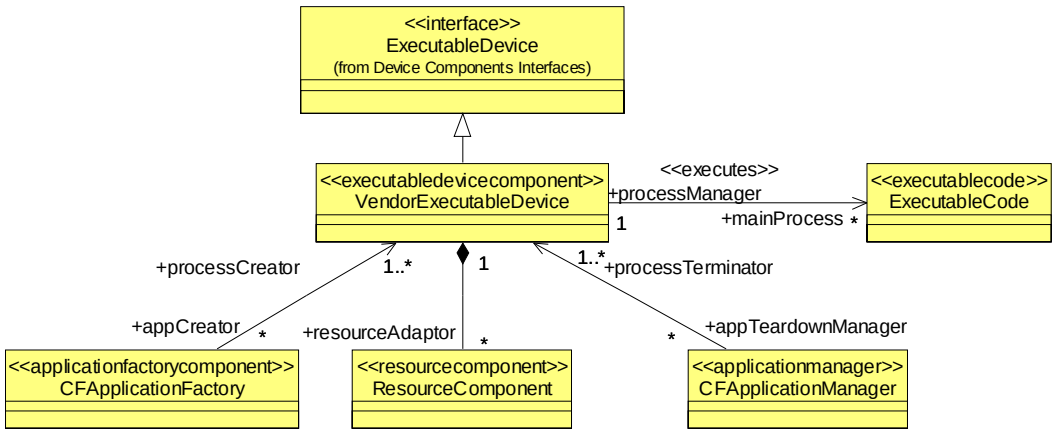

Figure 7.12 - ExecutableDeviceComponent M1 Illustration

M1 Associations

- `mainProcess: ExecutableCode [*]`
  Zero to many MainProcesses may be executed and terminated on a device.
- `resourceAdaptor: ResourceComponent [*]`
  A ResourceComponent can take on the role of an adaptor that communicates with components that are not implemented with a distributive component middleware environment.

Constraints

The ExecutableDeviceComponent shall realize the ExecutableDevice interface.

7.1.6.2.5 LoadableDeviceComponent

Description

The LoadableDeviceComponent, as shown in Figure 7.13, extends the DeviceComponent component by adding software loading and unloading behavior.

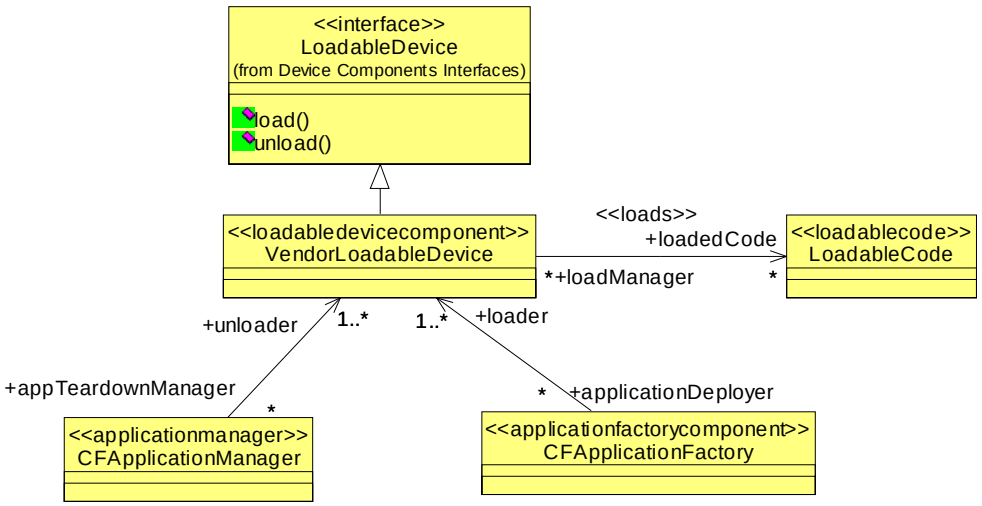

Figure 7.13 - LoadableDeviceComponent M1 Illustration

Attributes

- `<<characteristicselectionproperty>> loadKind: LoadType[1..*]`
  The <<characteristicselectionproperty>> (Application and Device Components::Properties) LoadKind defines the type of LoadTypes supported for the LoadableDevice load operation. Valid values are: DLL, DRIVER, EXECUTABLE, KERNEL MODULE, and SHARED LIBRARY. The valid values are device specific.
- `<<characteristicsetproperty>> os: NameVersionCharacteristic[1..*]`
  The <<characteristicsetproperty>> (Application and Device Components::Properties) os defines the type of Operating Systems supported for the LoadableDevice load operation. The OS name values are case sensitive. The value attributes are device specific.
- `<<characteristicsetproperty>>runtime: NameVersionCharacteristic[*]`
  The <<characteristicsetproperty>> (Application and Device Components::Properties) runtime defines the runtime environments supported for the LoadableDevice load operation. The value attributes are device specific.
- `<<characteristicsetproperty>>library: NameVersionCharacteristic[*]`
  The <<characteristicsetproperty>> (Application and Device Components::Properties) library defines the libraries supported for the LoadableDevice load operation. The value attributes are device specific.

M1 Associations

- `ObjectCode: ObjectCode [*]`
  Zero to many ObjectCodes may be loaded on a device.

Constraints

The LoadableDeviceComponent shall support the load type capabilities identified in the Device's component descriptor as specified in the softwareProfile attribute.

When a LoadKind characteristic property is not defined for the LoadableDeviceComponent, the load operation shall support all load kinds.

The LoadableDeviceComponent shall realize the LoadableDevice interface.

Semantics

The loaded software may be subsequently executed on the LoadableDeviceComponent, if the component is also an ExecutableDeviceComponent.

7.1.7 Application Components

This section defines the types (7.1.7.1) and component stereotypes (7.1.7.2) for application components.

7.1.7.1 Application Components Types

The Application Component Types define the types used by some of the Application stereotypes definitions.

Types and Exceptions

- `DomainFinderType(name: String, type: String)`
  The DomainFinderType is used to specify information about a ServiceComponent that a component such as DomainManagerComponent is aware of. The name attribute indicates the name of the ServiceComponent and type attribute indicates the type of ServiceComponent (e.g., log service, event service, etc.).
- `ResourceFactoryType(resourceFactoryInstantiationRef: String, resourceFactoryCreateOptions: ResourceFactoryComponent)`
  The ResourceFactoryType, a dataType, specifies the ResourceFactoryComponent instantiation element to use to obtain a ComponentInstantiation reference by the resourceFactoryInstantiationRef. The resourceFactoryCreateOptions are the options parameter values to be used for the ResourceFactoryComponent create operation.

7.1.7.2 Application Components Stereotypes

The Application Components Stereotypes section defines the set of components used to define applications and waveforms. The Application Components stereotypes are depicted in Table 7.5, which are extensions of the UML Component (UML2.0::Components::BasicComponents). The following subsections describe the details of these elements.

Table 7.5 - Applications Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
Application	Component	N/A		See constraints in section below	Represents an assembly of ApplicationResources and Components.
Application Resource Component	Component	Resource Component		See constraints in section below	Provides a common definition for an application's ResourceComponent.
Component Instantiation	Property	N/A	assembler Controller, hostCollocation, namingService, process Collocation resourcefactory,	See constraints in section below	Represents an Application Part.
LinkLayer ControlResource	Component	WaveformLayer Resource		See constraints in section below	Represents a standard Link Layer Control component of the OSI layer model.
MediumAccess ControlResource	Component	WaveformLayer Resource		See constraints in section below	Represents a standard Medium Access Control component of the OSI layer model.
NeworkLayer Resource	Component	WaveformLayer Resource		See constraints in section below	Represents a standard Network Layer component of the OSI layer model.
PhysicalLayer Resource	Component	WaveformLayer Resource		See constraints in section below	Represents a standard Physical Layer component of the OSI layer model.
ServiceConnector	Connector	N/A	deviceThatLoaded Component Instantiation, domainfinder, namingService, serviceComponent UsedBy Component Instantiation	See constraints in section below	Represents a connector to a service, interface or ServiceComponent.
ServiceComponent Instatiation	Property	N/A		See constraints in section below	Represents a service component within an Application.
Waveform Application	Component	Application			Represents a waveform application.
WaveformLayer Resource	Component	Application Resource Component			Represents a standard component of the OSI layer model.

7.1.7.2.1 Application

Description

The Application, as shown in Figure 7.14, provides a component assembly definition for a set of ApplicationResourceComponent(s) and Component(s).

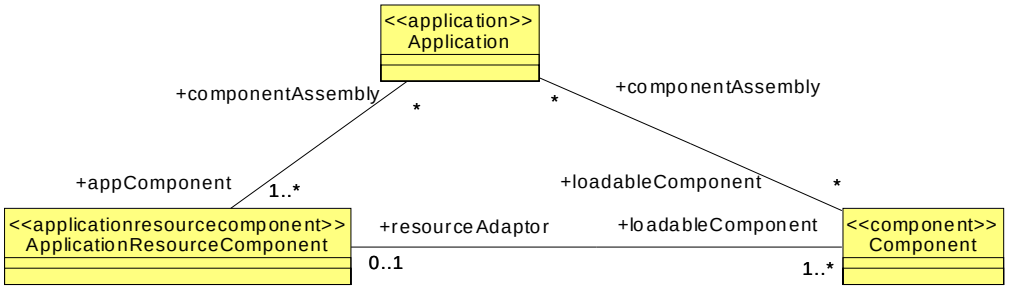

Figure 7.14 - Application M1 Illustration

M1 Associations

- `appComponent: ApplicationResource [1..*]`
  The set of ApplicationResources that are connected together to form the application assembly.
- `loadableComponent: Component [*]`
  The set of signal processing components that comprise the application assembly.

Constraints

The Application needs one ComponentInstantiation to be noted as an assembly controller.

The Application shall only be composed of ComponentInstantiations and ServiceComponentInstantiations.

The Application shall have at least one ComponentInstantiation and its type is other than a ResourceFactoryComponent type.

A ComponentInstantiation's resourceFactory resourceFactoryCreateOptions attribute shall have an instance value that agrees with the attributes for the referenced ResourceFactoryComponent ComponentInstantiation's type.

A ComponentInstantiation's resourceFactory resourceFactoryInstantiationRef shall be to a valid ResourceFactoryComponent ComponentInstantiation reference within the Application.

A ServiceConnector is not depicted between ComponentInstantiations' connections.

7.1.7.2.2 ApplicationResourceComponent

Description

The ApplicationResourceComponent, a specialization of ResourceComponents shown in Table 7.5, provides a common API for control and configuration of an application resource component. Figure 7.15 depicts the associations for an ApplicationResourceComponent.

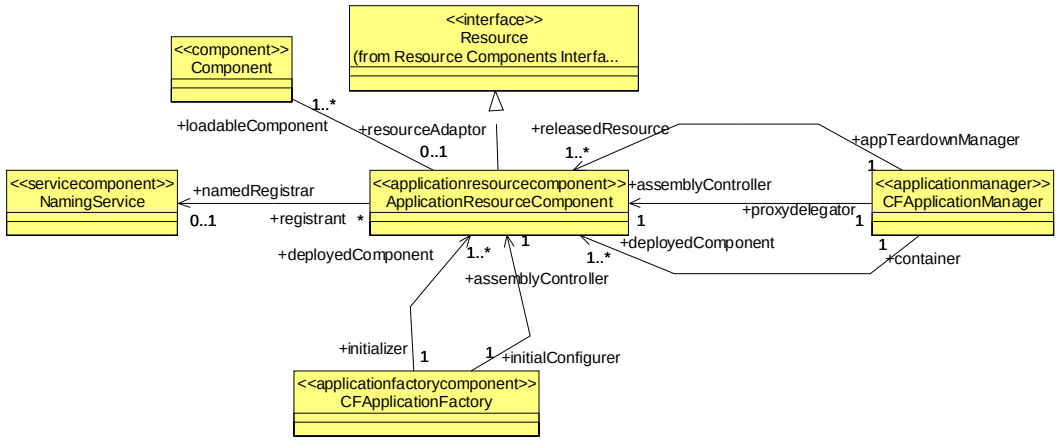

Figure 7.15 - ApplicationResourceComponent M1 Illustration

M1 Associations

- `namedRegistrar: NamingService [0..1]`
  The optional NamingService that contains a named ApplicationComponent reference.
- `loadableComponent: Component [*]`
  A loadableComponent may be associated with an ApplicationResourceComponent when the component acts as a resourceAdaptor. In this case, the loabableComponent cannot be communicated with unless through the resourceAdaptor.

Constraints

An ApplicationResourceComponent shall be registered with a NamingService when a ResourceFactoryComponent does not create the ApplicationResourceComponent.

Semantics

ApplicationResourceComponent references are contained in NamingService so the deployment machinery (e.g., ApplicationFactory) can obtain the deployed component.

7.1.7.2.3 ComponentInstantiation

Description

The ComponentInstantiation, an extension of Property, defines a specific component type for an Application's part.

Attributes

- `assemblyController: Boolean = False`
  The assemblyController indicates whether or not the ComponentInstantiation is acting as the assembly controller for the assembly composition. A value of True means the ComponentInstantiation is an assembly controller.
- `hostCollocation: String [0..1]`
  The hostCollocation attribute identifies the requirement for a ComponentInstantiation on being deployed on the same device with other ComponentInstantiations that have the same hostCollocation attribute value.
- `processCollocation: String [0..1]`
  The processCollocation attribute identifies the requirement for a ComponentInstantiation on being deployed within the same process space with other componentInstantiations that have the same processCollocation attribute value.
- `namingService: String [0..1]`
  The namingService attribute specifies the simple name used by the deployment machinery such as ApplicationFactoryComponent in forming the naming context for where the ComponentInstantiation reference is placed in a naming service.
- `resourceFactory: ResourceFactoryType [0..1]`
  The resourceFactory attributes indicates to the deployment machinery such as ApplicationFactoryComponent that the ComponentInstantiation is obtained by a ResourceFactoryComponent.

Constraints

The ComponentInstantiation's assemblyController attribute value shall only be True when the ComponentInstantiation type is a ResourceComponent or Component type.

The ComponentInstantiation's type shall be a ResourceFactoryComponent, ResourceComponent, Component, or Application.

The ComponentInstantiation shall have a value for either namingService or resourceFactory attribute.

The ComponentInstantiation's value shall be an instance value that agrees with the attributes for the ComponentInstantiation's type.

7.1.7.2.4 LinkLayerControlResource

Description

The LinkLayerControlResource, as shown in Table 7.5, specializes the WaveformLayerResource stereotype. The LinkLayerControlResource stereotype provides a mechanism to realize a standard Link Layer Control component of the OSI layer model. A LinkLayerControlResource is associated with a physical layer resource or a medium access control resource as shown in Figure 7.16. The standard facilities of a Link Layer Control API are defined in reference 3.2.3.

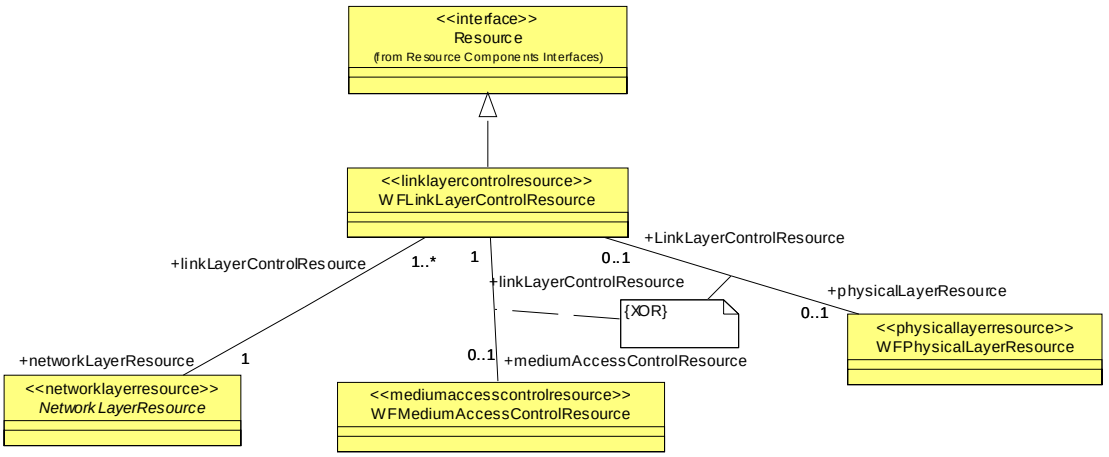

Figure 7.16 - LinkLayerControlResource M1 Illustration

M1 Associations

- `physicalLayerResource : WFPhysicalLayerResource [0..1]`
  LinkLayerControlResource may communicate directly with PhysicalLayerResource, by-passing the Medium Access layer. This communication is only in the control plane. In other scenarios where a Medium Access layer is not present in the waveform, this association encompasses both control and data plane communication between those two components. (Vertical Communication)
- `mediumAccessControlResource : WFMediumAccessControlResource [0..1]`
  LinkLayerControlResource communicates with MediumAccessControlResource, and LinkLayerControlResource controls the transmission parameters related to the link establishment and quality. MediumAccessControlResource may perform some quality of service related measurements and communicate this to the Link controller. Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)
- networkLayerResource: NetworkLayerResource [1]
  NetworkLayerResource communicates with LinkLayerControlResources.

Constraints

The LinkLayerControlResource shall be associated with a NetworkLayerResource.

The LinkLayerControlResource shall be either associated with a PhysicalLayerResource or a MediumAccessControlResource.

7.1.7.2.5 MediumAccessControlResource

Description

The MediumAccessControlResource, as shown in Table 7.5, specializes the WaveformLayerResource stereotype. The MediumAccessControlResource stereotype provides a mechanism to realize a standard Medium Access Control component of the OSI layer model. A MediumAccessControlResource is associated with a physical layer resource and link layer control resource as shown in Figure 7.17. The standard facilities of a Medium Access Control API are defined in reference 3.2.3.

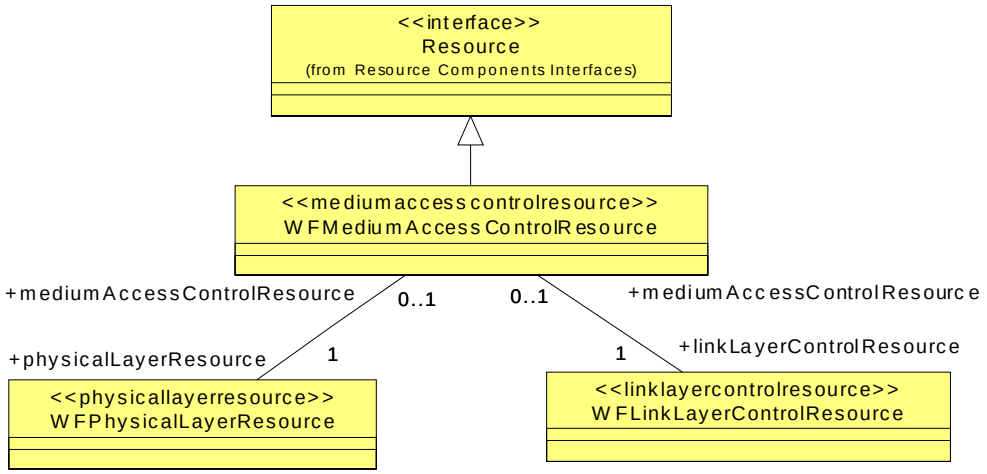

Figure 7.17 - MediumAccessControlResource M1 Illustration

M1 Associations

- `physicalLayerResource : WFPhysicalLayerResource [1]`
  PhysicalLayerResource communicates with MediumAccessControlResource, and MediumAccessControlResource controls the transmission parameters related to the physical medium. Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)
- `linkLayerControllerResource : WFLinkLayerControllerResource [1]`
  MediumAccessControlResource communicates with LinkLayerControllerResource, and LinkLayerControllerResource controls the transmission parameters related to the link establishment and quality. MediumAccessController Resource may perform some quality of service related measurements and communicate this to the Link controller. Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)

Constraints

A MediumAccessControlResource shall be associated with a PhysicalLayerResource and a LinkLayerControlResource.

7.1.7.2.6 NetworkLayerResource

Description

The NetworkLayerResource, as shown in Figure 7.18, specializes the WaveformLayerResource stereotype. The NetworkLayerResource stereotype provides a mechanism to realize a standard Network Layer component of the OSI layer model. A NetworkLayerResource is associated with one or more link layer control resources and can also play the role of a gateway/translator as shown in Figure 7.18. The facilities of a network layer component is out of the scope of this specification and is not included in the PIM.

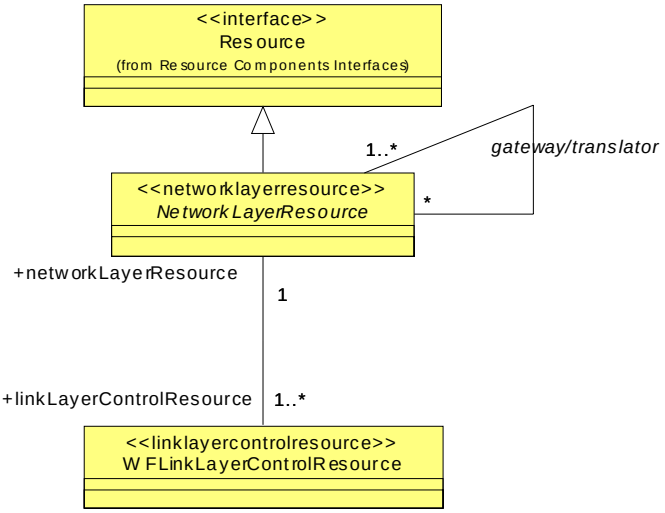

Figure 7.18 - NetworkLayerResource M1 Illustration

Associations

- `linkLayerControlResource : WFLinkLayerControlResource [1..*]`
  A NetworkLayerResource may communicate with one or more Link Layer components within the waveform.
- `Gateway/translator: NetworkLayerResource */[1..*]`
  A platform may be programmed to act as a waveform bridge / repeater, and in the case the network layer resource communicates with other network layer resources to provide gateway/waveform translator functionality. (Horizontal Communication).

Constraints

A NeworkLayerResource shall be associated with one or more LinkLayerControlResources.

7.1.7.2.7 PhysicalLayerResource

Description

The PhysicalLayerResource, as shown in Table 7.5, specializes the WaveformLayerResource stereotype. The PhysicalLayerResource stereotype provides a mechanism to realize a standard Physical Layer component of the OSI layer model. A PhysicalLayerResource is associated with a medium access control or link layer control resource as shown in Figure 7.19. The standard facilities of a Physical Layer API are defined in references 3.2.3 and 3.2.4.

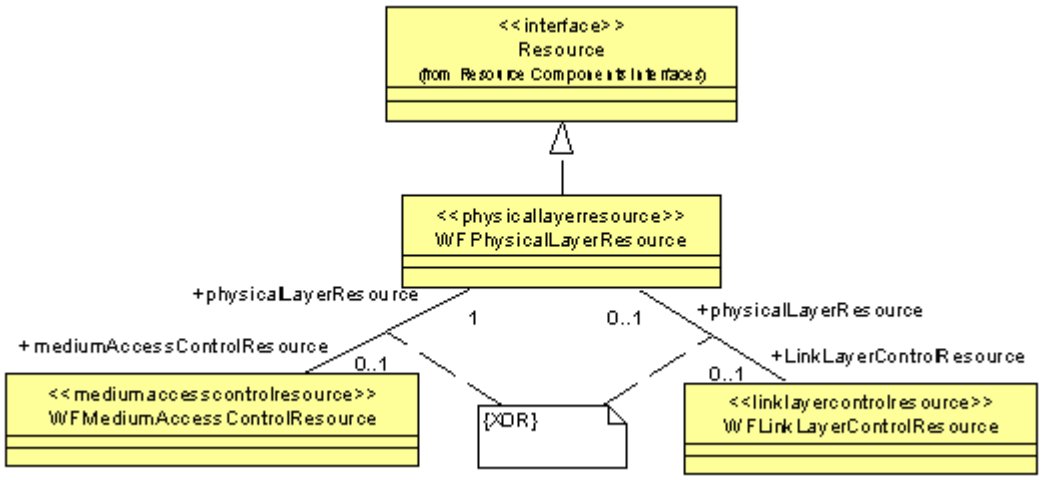

Figure 7.19 - PhysicalLayerResource M1 Illustration

M1 Associations

- `MediumAccessControlResource : WFMediumAccessControlResource [0..1]` PhysicalLayerResource communicates with MediumAccessControlResource, and MediumAccessControlResource controls the transmission parameters related to the physical medium. Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)
- `linkLayerControllerResource : WFLinkLayerControllerResource [0..1]` PhysicalLayerResource communicates with LinkLayerControllerResource, and LinkLayerControllerResource controls the transmission parameters related to the link establishment and quality. Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)

Constraints

A `PhysicalLayerResource` shall be associated with either a `MediumAccessControlResource` or a `LinkLayerControlResource`.

7.1.7.2.8 ServiceComponentInstantiation

Description

The ServiceComponentInstantiation, an extension of property, defines a ServiceComponent type for an Application's part so one can show ComponentInstantiations connections to ServiceComponents. These components are not physical part of the Application but are platform ServiceComponents used by ComponentInstantiations.

Constraints

The ServiceComponentInstantiation's type shall be a ServiceComponent stereotype.

7.1.7.2.9 ServiceConnector

Description

The serviceconnector specifies to the deployment machinery such as ApplicationFactoryComponent on how to obtain the ServiceComponent reference that is participating in the connections. ServiceConnector is used for connection to platform ServiceComponents and their interfaces that an Application's ComponentInstantiation is connected to.

Attributes

- `deviceThatLoadedComponentInstantiation: String [0..1]`
- `DomainFinder: DomainFinderType [0..1]`
- `namingService: String [0..1]`
  The namingService attribute specifies a full naming context that identifies the location of the ServiceComponent reference in a naming service.
- `serviceUsedByComponentInstantiation: ServiceComponent [0..1]`

Constraints

A ServiceConnector needs to have a value specified for one of its attributes.

7.1.7.2.10 WaveFormLayerResource

Description

The WaveformLayerResource, as shown in Table 7.5, specializes the ApplicationResource stereotype. The WaveformLayerResource stereotype provides a mechanism to realize a waveform layer component, should a non-OSI layer be required. For standard OSI layers, the stereotypes that specialize the WaveformLayerResource should be used.

A WaveformLayerResource can perform two types of communication as shown in Figure 7.20. In the **horizontal communication** scenario, a waveform layer component communicates with one or more peer waveform layer components that are located in another platform. In the **vertical communication** scenario, a waveform component communicates with other waveform components within the same platform.

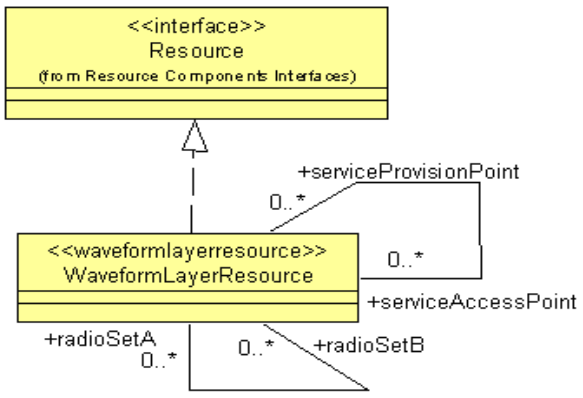

Figure 7.20 - WaveformLayerResource M1 Illustration

M1 Associations

- `DomainA, DomainB: WaveformLayerResource [0..*]`
  This association shows the data transfer between two WaveformLayerResources that reside in different platforms. This is an example of horizontal communication.

- `serviceProvisionPoint, serviceAccessPoint: WaveformLayerResource [0..*]`
  This association shows the data transfer between two or more WaveformLayerResources that are in the same platforms. This is an example of vertical communication. Service Access Point is defined as an interface or a port on the client waveform component through which the client uses a service. Service Provision Point is defined as a port or interface on a server providing a service.

7.2 Infrastructure

7.2.1 Services

This section defines the interfaces and stereotypes for services. The following subsections provide the definitions for Services Interfaces, Services Stereotypes, and File Services.

7.2.1.1 Services Interfaces

The following subsections provide the definitions for StateManagement and CapabilityModel(s).

7.2.1.1.1 CapabilityModel

Description

The abstract CapabilityModel (Figure 7.21) provides the ability to determine if a ServiceProperty can satisfy a deployment requirement. As shown in Figure 7.21, there are two specializations of a CapabilityModel, which are: CharacteristicModel and CapacityModel.

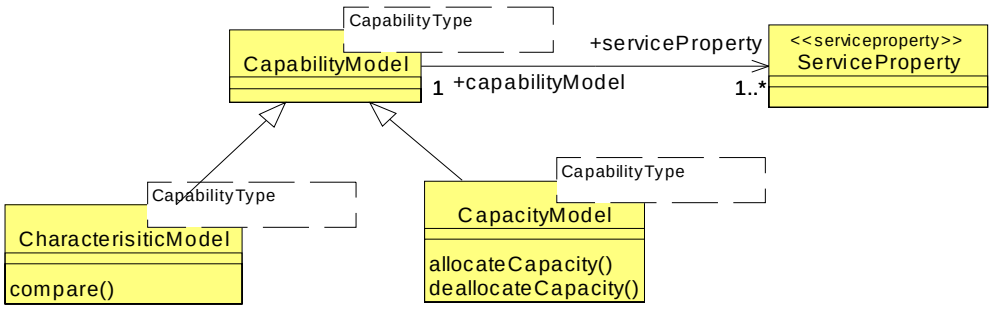

Figure 7.21 - CapabilityModels Definition

M1 Associations

- `serviceProperty: ServiceProperty [1..n]`
  A ServiceProperty(s) is associated with a CapabilityModel that determines the feasibility of the ServiceProperty(s) satisfying a deployment requirement.

Semantics

A CapabilityModel may be applicable for a set of ServiceProperty(s) depending on the ServiceProperty(s)' value type. For example, a CapabilityModel that compares string types may be applicable for a number of ServiceProperty(s) that are of a String type. CapabilityModel(s) are implemented or used by the deployment machinery such as ApplicationFactory(s) and ManagedServiceComponent(s).

7.2.1.1.2 CapacityModel

Description

The CapacityModel, as shown in Figure 7.21, provides the ability to manage allocation and deallocation of a CapacityProperty.

Operations

- `allocateCapacity (in requiredCapacity: CapabilityType, return Boolean)`
  The allocateCapacity operation provides the mechanism to request and allocate capacity. The allocateCapacity operation behavior is implementation specific.
- `deallocateCapacity (in requiredCapacity: CapabilityType)`
  The deallocateCapacity operation provides the mechanism to deallocate capacity. The deallocateCapacity operation behavior is implementation specific.

Constraints

A CapacityModel shall be associated with CapacityProperty.

Semantics

The types of CapacityProperty managed by a CapacityModel are usually numeric types. Examples of CapacityModel(s) are: decrement and increment counter types and quantity subtraction and addition types.

7.2.1.1.3 CharacteristicModel

Description

The CharacteristicModel, as shown in Figure 7.21, provides the ability to determine the feasibility of a characteristic property that is static characteristic property.

Operations

- `compare (in requiredcharacteristic: CapabilityType, return Boolean)`
  The compare operation provides a generic compare mechanism to determine whether a characteristic can satisfy a requiredcharacteristic request. The implementation of the compare operation is implementation specific.

Constraints

A CharacteristicModel shall be associated with characteristic property type.

Semantics

The types of characteristic property compared by a CharacteristicModel can be any type (e.g., CharacteristicProperty, CharacteristicSelectionProperty, and CharacteristicSetProperty). Examples of types of CharacteristicModel(s) are equality operators: “eq,” “n,e” “le,” “ge,” “gt,” and “lt” and also selection/matching operators.

7.2.1.1.4 StateManagement

Description

The StateManagement interface, as shown in Figure 7.22, provides the ability to retrieve state information and administratively manage a component. The StateManagement interface incorporates aspects of the Administrative, Operational, and Usage states described in ISO/IEC International Standard 10164-2.

Attributes

- `<<readonly>>adminState: AdminType [0..1]`
  The administrative state indicates the permission to use or prohibition against using the component. The adminState attribute contains the admin state value.

- `<<readonly>>operationalState : OperationalType`
  The readonly operationalState attribute contains the component’s operational state (ENABLED or DISABLED). The operational state indicates whether or not a component is functioning.

- `<<readonly>>adminStateRequestSupportedCharacteristic : AdminRequestSupportedType = UNLOCK`
  The adminStateRequestSupportedCharacteristic attribute indicates the behavior requirement for the StateManagement’s setAdminState operation and adminState attribute, which indirectly affect the admin states that have to be supported.

- `<<readonly>>states: StatesType`
  The readonly states attribute contains the values for all three states: admin, operational, and usage.

- `<<readonly>>usageState: UsageType`
  The readonly usageState attribute contains the component’s usage state (IDLE, ACTIVE, or BUSY). UsageState indicates whether or not a component is in use, and if so, whether or not it has spare capacity for allocation.

Operations

- `setAdminState (in adminRequest: AdminRequestType): {raises = ( UnsupportedRequest )}`
  The setAdminState operation changes the adminState attribute based upon the input adminRequest parameter.

  The setAdminState operation shall set the adminState attribute to UNLOCKED upon receipt of an UNLOCK admin request.

  The setAdminState operation shall set the adminState to LOCKED and usageState to IDLE upon receipt of a LOCK admin request, when the adminStateRequestSupportedCharacteristic attribute value is LOCK or ALL.

  The setAdminState operation shall set the adminState to SHUTTING_DOWN upon receipt of a SHUTDOWN admin request, when the adminState attribute is UNLOCKED, and adminStateRequestSupportedCharacteristic attribute value is SHUTDOWN or ALL.

The setAdminState operation shall raise the UnsupportedRequest exception when the adminRequest is not supported.

Types and Exceptions

- `<<enumeration>>AdminRequestType (LOCK, SHUTDOWN, UNLOCK)`
  The AdminRequestType defines the administrative state request values for a component as follows:
  - LOCK - requests a forceful state transition to the LOCKED state.
  - SHUTDOWN - makes a graceful lock request that results in a transition to the SHUTTING_DOWN state.
  - UNLOCK - requests a transition to the UNLOCKED state.
- `<<enumeration>>AdminRequestSupportedType (ALL, NOT_IMPLEMENED, LOCK, SHUTDOWN, UNLOCK)`
  The AdminRequestSupportedType defines the valid administrative state request values for a component as follows:
  - ALL - all requests are supported
  - NOT_IMPLEMENTED - no requests are supported
  - LOCK - only LOCK and UNLOCK
  - SHUTDOWN - only SHUTDOWN and UNLOCK
  - UNLOCK - only UNLOCK
- `<<enumeration>>AdminType (LOCKED, SHUTTING_DOWN, UNLOCKED)`
  The AdminType defines the administrative state values for a component as follows:
  - LOCKED - The component is reserved for administrative usage only. For example, no capacity requests are granted by a component such as device. Transitions to this state may originate from either the SHUTTING_DOWN or UNLOCKED state.
  - SHUTTING_DOWN - a transition state from UNLOCKED to LOCKED, this state does not allow capacity requests to be granted successfully and is maintained until all capacities are deallocated and the component is not in use.
  - UNLOCKED - The component is available for full usage providing the operationalState is ENABLED. A state transition from SHUTTING_DOWN or LOCKED can occur.
- `<<enumeration>>OperationalType`
  The OperationalType defines the operational state values for a component as follows:
  - ENABLED - the component is functional
  - DISABLED - the component is non-functional
- `StatesType( adminState : AdminType, operationalState : OperationalType, usageState : UsageType)`
  The StatesType contains the state values for a component.
- `<<enumeration>>UsageType`
  The UsageType defines the usage state values for a component as follows:
  - IDLE - not in use
  - ACTIVE - in use, with capacity remaining for allocation, or
  - BUSY - in use, with no capacity remaining for allocation

Constraints

When the adminStateRequestSupportedCharacteristic value attribute is NOT_IMPLEMENTED, then a component that realizes this interface may not support setAdminState operation and adminState attribute. The NOT_IMPLEMENTED behavior is dependent of the PSM.

The adminState attribute shall transition to the LOCKED state from SHUTTING_DOWN state when the component's usageState attribute becomes IDLE and the adminStateRequestSupportedCharacteristic attribute value is SHUTDOWN or ALL.

Semantics

The interface is used by components that manage their own states and capacities, and provide administrative control. The relationships, values, and state transitions among the adminState, operationalState, usageState, and attributes are described in the ISO/IEC International Standard 10164-2.

7.2.1.2 Services Stereotypes

The service stereotypes, which are extensions of the UML 2.0 Component (UML2.0::Components::BasicComponents) and Interface (UML2.0::Classes::Interfaces) classifiers are depicted in Table 7.6.

Table 7.6 - Services Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
ManagedServiceComponent	Component	ServiceComponent		See constraints in section below	Offers Service(s) within the environment, which can be used by components.
Service	Interface	N/A			Provides the parent interface definition for the services within the environment that can be used by components.
ServiceComponent	Component	N/A			Provides the parent component definition for the service components within the environment that can be used by components.

7.2.1.2.1 ManagedServiceComponent

Description

The ManagedServiceComponent is a type of ServiceComponent that provides state management behavior as shown in Table 7.6 and Figure 7.22.

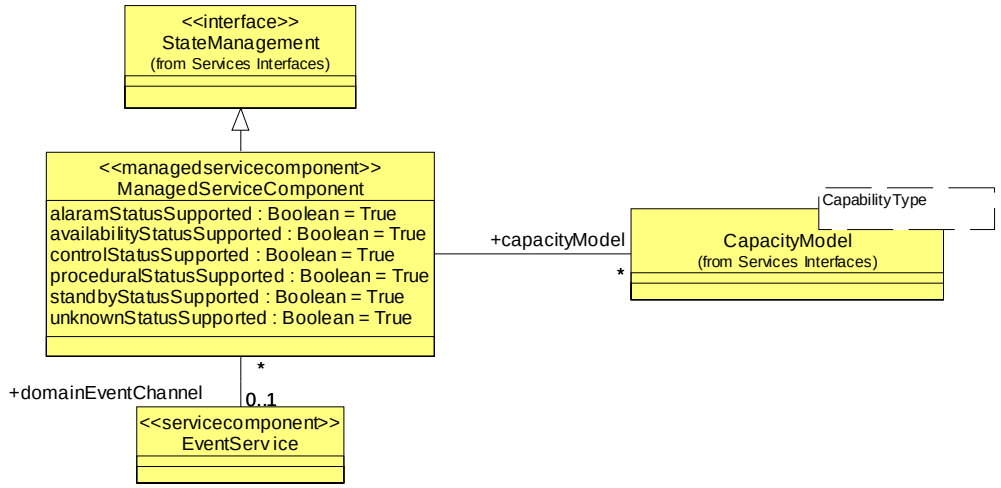

Figure 7.22 - ManagedServiceComponent M1 Illustration

M1 Associations

- `capacityModel: CapacityModel [*]`
  The capacityModel used by a managed service depends on the set of capacities managed by the component.
- `domainEventChannel: EventChannelService [0..1]`
  The event channel used by a managed service for indicating state changes.

Attributes

- `alarmStatusSupported: Boolean = false`
  The alarmStatusSupported attribute is used to indicate if the `<<configureproperty>>alarmStatus` is supported. A value of true means supported by the component. An alarmStatus can have zero or more of the following values, not all of which are applicable to every class of managed component. When the value of this attribute is empty set, this implies that none of the status conditions described below are present. AlarmStatus Ushort values are: under repair = 2, critical = 4, major = 8, minor = 16, and alarm outstanding = 32.
- `availabilityStatusSupported: Boolean = false`
  The availabilityStatusSupported attribute is used to indicate if the `<<queryproperty>>availabilityStatus` is supported. A value of true means supported by the component. The availabilityStatus attribute can have zero or more of the following values, not all of which are applicable to every class of managed component. When the value of this attribute is empty set, this implies that none of the status conditions described below are present. AvailabilityStatus UShort values are: in test = 2, failed = 4, power off = 8, off line = 16, off duty = 32, dependency = 64, degraded = 128, not installed = 256, and log full = 512.

- `controlStatusSupported: Boolean = false`
  The controlStatusSupported attribute is used to indicate if the `<<configureproperty>>controlStatus` is supported. A value of true means supported by the component. The controlStatus attribute that can have zero or more of the following values, not all of which are applicable to every class of managed component. When the value of this attribute is empty set, this implies that none of the status conditions described below are present. Control Status UShort values are: subject to test =2, part of services locked = 4, reserved for test = 8, and suspended = 16.

- `proceduralStatusSupported: Boolean = false`
  The proceduralStatusSupported attribute is used to indicate if the `<<queryproperty>>procedualStatus` is supported. A value of true means supported by the component. The proceduralStatus is supported only by those classes of managed components that represent some procedure (e.g., a test process) which progresses through a sequence of phases. Depending upon the managed component definition, the procedure may be required to reach certain phase for the resource to be operational and available for use (i.e., for the managed component to be enabled). Not all phases may be applicable to every class of managed component. If the value of this attribute is an empty set, the managed component is ready, for example, the initialization is complete. When the value of this attribute is empty set, this implies that none of the status conditions described below are present. ProceduralStatus UShort values are: initialized required = 2, not initialized = 4, initializing = 8, reporting = 16, and terminating = 32.

- `standbyStatusSupported: Boolean = false`
  The standbyStatusSupported attribute is used to indicate if the `<<queryproperty>>standbyStatus` is supported. A value of true means supported by the component. The standbyStatus attribute is single-valued and is only meaningful when the back-up relationship role exists. StandbyStatus UShort values are: standby hot = 2, standby cold = 4, and providing service = 8.

- `unknownStatusSupported: Boolean = false`
  The unknownStatusSupported attribute is used to indicate if the `<<queryproperty>>unknownStatus` is supported. A value of true means supported by the component. The unknownStatus attribute is used to indicate that the state of the resource represented by the managed component is unknown. When the unknownStatus attribute Boolean value is True, the value of the state attributes may not reflect the actual state of the resource.

Constraints

A ManagedServiceComponent shall at a minimum support the UNLOCKED adminState value.

A ManagedServiceComponent shall at a minimum support the ENABLED operationalState value.

A ManagedServiceComponent shall support the IDLE, and ACTIVE and/or BUSY usageState values. The supported usageState values depend on the component's usage model.

A ManagedServiceComponent shall be associated with at least one CapacityProperty as described by the component's descriptor.

When alarmStatusSupported attribute is true, then the ManagedServiceComponent shall support the configuring and querying of the alarmStatus property.

When availabilityStatusSupported attribute is true, then the ManagedServiceComponent shall support the querying of the availabilityStatus property.

When controlStatusSupported attribute is true, then the ManagedServiceComponent shall support the configuring and querying of the controlStatus property.

When proceduralStatusSupported attribute is true, then the ManagedServiceComponent shall support the querying of the proceduralStatus property.

When standbyStatusSupported attribute is true, then the ManagedServiceComponent shall support the querying of the standbyStatus property.

When unknownStatusSupported attribute is true, then the ManagedServiceComponent shall support the querying of the unknownStatus property.

Semantics

A ManagedServiceComponent may be associated with one to many ServiceProperty(s) (Characteristic Properties and CapacityProperty). The CapacityModel(s) associated with a ManagedServiceComponent depends on the CapacityProperty(s) managed by the component. The capacities associated with a ManagedServiceComponent may be managed or not managed by the component.

The adminStateRequestSupportedCharacteristic attribute may be dynamically set at creation by a ConfigureProperty or ExecutableProperty.

Whenever the adminState attribute changes, a StateChangeEventType (Infrastructure::Domain Management::Event Channels) event may be issued to an event channel. The StateChangeEventType event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the component.
2. The sourceId field is the identifier attribute of the component.
3. The stateChangeCategory field is ADMINISTRATIVE_STATE_EVENT.
4. The stateChangeFrom and stateChangeTo fields reflect the adminState attribute value before and after the state change, respectively.

Whenever the operationalState attribute changes, a StateChangeEventType event may be issued to an event channel. The event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the component.
2. The sourceId field is the identifier attribute of the component.
3. The stateChangeCategory field is OPERATIONAL_STATE_EVENT.
4. The stateChangeFrom and stateChangeTo fields reflect the operationalState attribute value before and after the state change, respectively.

Whenever the usageState attribute changes, a StateChangeEventType event may be issued to an event channel. The StateChangeEventType event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the component.
2. The sourceId field is the identifier attribute of the component.
3. The stateChangeCategory field is USAGE_STATE_EVENT.
4. The stateChangeFrom and stateChangeTo fields reflect the usageState attribute value before and after the state change, respectively.

7.2.1.2.2 ServiceComponent

Description

The ServiceComponent, as shown in Figure 7.23, offers service(s) within the environment, which can be used by Component(s). For example, the potential Service(s) offered by a ServiceComponent are log, event, and naming.

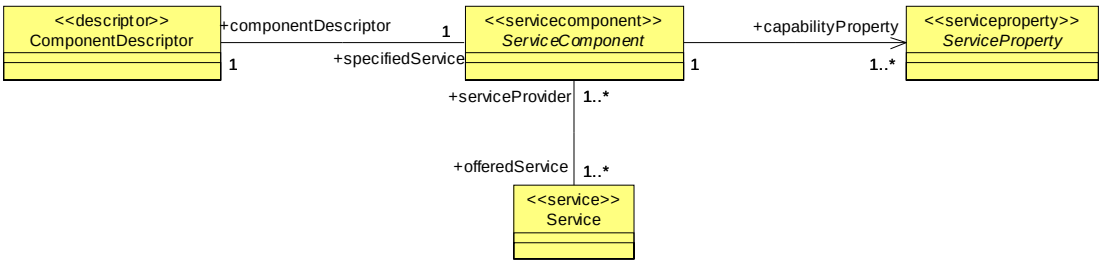

Figure 7.23 - ServiceComponent M1 Illustration

M1 Associations

- `componentDescriptor: ComponentDescriptor [1]`
  The Descriptor that provides the ServiceComponent definition.
- `capabilityProperty: ServiceProperty [1..*]`
  The capability properties that describe a Service's capabilities.
- `offeredService: Service [1..*]`
  An offeredService holds the set of Services that are provided by a ServiceComponent.

Semantics

ServiceComponent(s) are registered with DeviceManager(s). A DomainManagerComponent knows the ServiceComponent(s) when DeviceManagerComponent(s) registers to a DomainManagerComponent.

7.2.1.3 File Services

The FileServices consist of interfaces and components that are used to manage and access a distributed file system. The File Services are used for installation and removal of application and artifact files within the system, and for loading and unloading those files on the various processors that they execute upon. The File Services interfaces are described in Section 7.2.1.3.1 and the file services component stereotypes are described in Section 7.2.1.3.2. The relationships between the interfaces and components are graphically depicted in Figure 7.24.

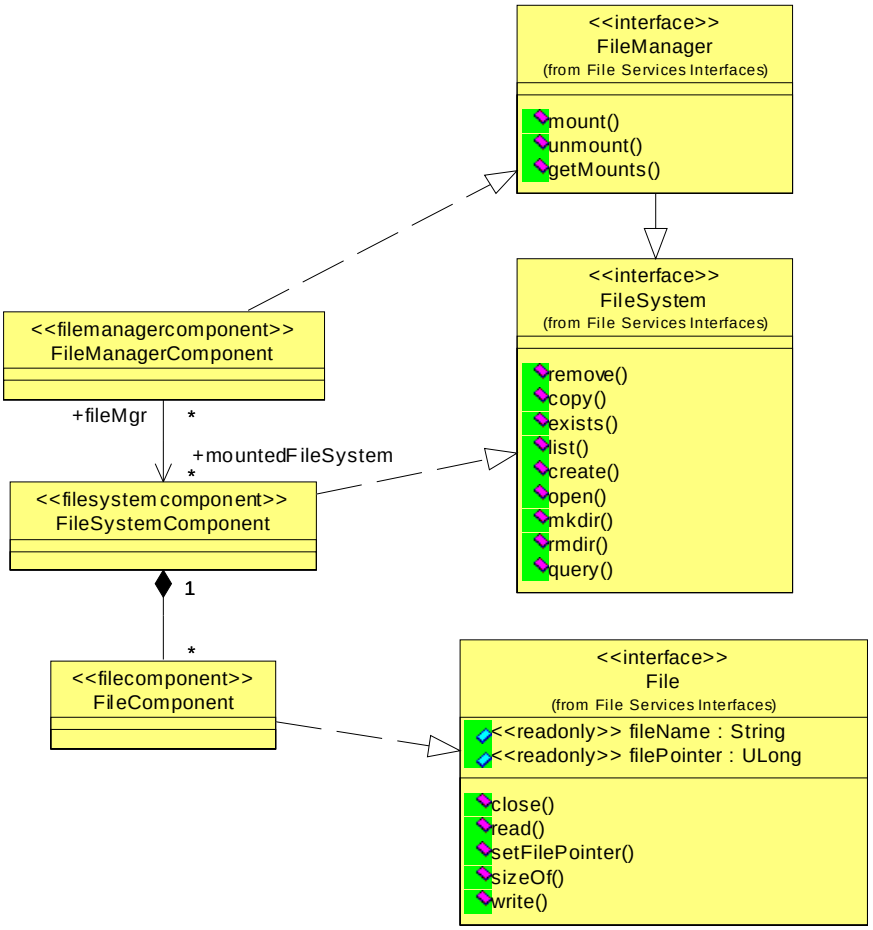

Figure 7.24 - File Services Overview

7.2.1.3.1 File Services Interfaces

This section defines the interfaces for File, FileSystem, and FileManager as shown in Figure 7.24. These interfaces provide basic file manipulation operations that are described in detail in the following subsections.

Types and Exceptions

- `<<exception>>FileException`
  The FileException, is a type of SystemException, that indicates a file-related error. The error number indicates an ErrorNumberType value (e.g., CF_EBADF, CF_EEXIST, CF_EISDIR, CF_EMFILE, CF_ENFILE, CF_ENOENT, CF_ENOSPC, CF_ENOTDIR, CF_ENOTEMPTY, CF_EROFS). The String msg attribute can provide information describing why the FileException occurred.

7.2.1.3.1.1 File

Description

The File interface, as shown in Figure 7.24, provides the ability to read and write files residing within a potentially distributed FileSystem. A file can be thought of conceptually as a sequence of Octet(s) with a current filePointer describing where the next read or write will occur. This filePointer points to the beginning of the file upon construction of the file object.

Attributes

- `<<readonly>>fileName: String`
  The readonly fileName attributes shall contain the file name given to a file system open or create.
- `<<readonly>>filePointer: ULong`
  The readonly filePointer attribute shall contain the file position where the next read or write will occur.

Operations

- `close(): {raises= ( FileException)};`
  The close operation shall close the file from the file system. The close operation shall remove the File component. The close operation shall raise the FileException when it cannot successfully close the file.
- `read(out data : OctetSequence, in length : ULong) : {raises= ( IOException)};`
  The read operation shall read the number of Octet(s) specified by the input length parameter and advance the value of the filePointer attribute by the number of Octet(s) read. The read operation will read less than the number of octets specified if an end of file is encountered before the input length number of octets is read.

  The read operation shall return via the out data parameter an OctetSequence that equals the number of octets actually read from the File. If the filePointer attribute value is at the end of the File prior to a read, the read operation shall return a 0-length OctetSequence. The read operation shall raise the IOException when a read error occurs.
- `setFilePointer(in filePointer : ULong) : {raises ( InvalidFilePointer, FileException )}`
  The setFilePointer operation shall set the filePointer attribute value to the input filePointer. The setFilePointer operation shall raise the FileException when the file pointer for the referenced file cannot be set to the value of the input filePointer parameter. The setFilePointer operation shall raise the InvalidFilePointer exception when the value of the filePointer parameter exceeds the file size.
- `sizeOf( return ULong) : }raises ( FileException )}`
  The sizeOf operation shall return the number of octets stored in the file. The sizeOf operation shall raise the FileException when a file-related error occurs (e.g., file does not exist anymore).
- `write(in data : OctetSequence) : {raises ( IOException)}`
  The write operation shall write data to the file. If the write is successful, the write operation shall update the filePointer attribute to reflect the number of octets written. If the write is unsuccessful, the filePointer attribute value shall maintain or be restored to its value prior to the write operation call. The write operation shall raise the IOException when a write error occurs.

Types and Exceptions

- `<<exception>>InvalidFilePointer`
  The InvalidFilePointer exception indicates the file pointer is out of range based upon the current file size.

- `<<exception>>IOException`
  The IOException exception, specialization of SystemException, indicates an error occurred during a read or writes operation to a File. The error number (e.g., CF_EFBIG, CF_ENOSPC, CF_EROFS and message is component-dependent. The message provides additional information describing the reason for the error.

Constraints

The filePointer shall be set to the beginning of the file when a File is opened for read only or created for the first time. When a File already exists and is opened for write, the filePointer shall be set at the end of the File.

7.2.1.3.1.2 FileManager

Description

The FileManager, as shown in Figure 7.24, specializes the FileSystem interface and extends that interface by adding mount and unmount operations. This allows multiple, distributed FileSystems to be accessed through a FileManager. The FileManager appears as a single FileSystem although the actual file storage may span multiple physical file systems. This is called a federated file system. A federated file system is created using the mount and unmount operations.

Operations

- `getMounts( return MountSequence)`
  The getMounts operation shall return a sequence of Mount structures that describe the mounted FileSystems.
- `mount(in string mountPoint, in FileSystemComponent file_System) : {raises( InvalidFileName, InvalidFileSystem, MountPointAlreadyExists )}`
  The mount operation shall associate the specified FileSystem with the given mountPoint. A mountPoint name shall begin with a "/." A mountPoint name is a logical directory name for a FileSystem. The mount operation shall raise the InvalidFileName exception when the input file name is invalid. The mount operation shall raise the MountPointAlreadyExists exception when the mountPoint already exists in the file manager. The mount operation shall raise the InvalidFileSystem exception when the input FileSystem is a null object reference.
- `query(inout fileSystemProperties : Properties) : {raises( UnknownFileSystemProperties )}`
  The query operation shall return the combined mounted file systems information to the calling client based upon the given input fileSystemProperties' IDs. As a minimum, the query operation shall support the following input fileSystemProperties IDs:
  - SIZE - a property item ID value of "SIZE" will cause the query operation to return the combined total size of all the mounted file system as an unsigned long long property value.
  - AVAILABLE_SPACE - a property item ID value of "AVAILABLE_SPACE" will cause the query operation to return the combined total available space (in Octet(s)) of all the mounted file system as unsigned long long property value.
- `unmount(in string mountPoint) : {raises( NonExistentMount )}`
  The unmount operation shall remove a mounted FileSystemComponent from the FileManagerComponent whose mounted name matches the input mountPoint name. The unmount operation shall raise the NonExistentMount exception when the mountPoint does not exist.

Types and Exceptions

- `MountType ( MountPoint : String, fs : FileSystemComponent)`
  The MountType structure identifies the FileSystems mounted within the FileManager.
- `MountSequence`
  The MountSequence is an unbounded sequence of MountTypes.

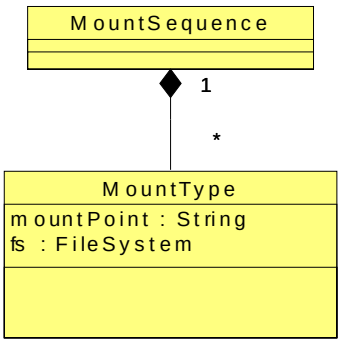

Figure 7.25 - MountSequence

- `<<exception>>InvalidFileSystem`
  The InvalidFileSystem exception indicates the FileSystemComponent is a null (nil) component reference.
- `<<exception>>MountPointAlreadyExists`
  The MountPointAlreadyExists exception indicates the mount point is already in use in the FileManagerComponent.
- `<<exception>>NonExistentMount`
  The NonExistentMount exception indicates a mount point does not exist within the FileManagerComponent.

Constraints

The FileManager's operations shall remove the FileSystem mounted name from the input fileName before passing the fileName to an operation on a mounted FileSystem.

The FileManager shall use the mounted FileSystem operations based upon the mounted FileSystem name that exactly matches the input fileName to the lowest matching subdirectory.

The FileManager shall propagate exceptions raised by a mounted FileSystem's operation.

Semantics

The FileManager's FileSystem operations behavior implements the requirements of the FileSystem operations against the mounted file systems. The FileManager's FileSystem operations ensure that the filename/directory arguments given are absolute pathnames relative to a mounted FileSystem.

The system may support multiple FileSystem implementations. Some FileSystems will correspond directly to a physical file system within the system. The FileManager supports a federated, or distributed, file system that may span multiple file system components. From the client perspective, the File Manager may be used just like any other FileSystem component since the FileManager realizes the FileSystem interface.

Based upon the pathname of a directory or file and the set of mounted FileSystems, the FileManager will delegate the FileSystem operations to the appropriate FileSystem. For example, if a FileSystem is mounted at /ppc2, an open operation for a file called /ppc2/profile.xml would be delegated to the mounted FileSystem. The mounted FileSystem will be given the filename relative to it. In this example the FileSystem's open operation would receive /profile.xml as the fileName argument.

Another example of this concept can be shown using the copy operation. When a client invokes the copy operation, the FileManager will delegate operations to the appropriate FileSystems (based upon supplied pathnames) thereby allowing copy of files between FileSystems.

7.2.1.3.1.3 FileSystem

Description

The FileSystem interface, as shown in Figure 7.24, defines common operations that enable access to a physical file system.

Attributes

- `<<constant>>SIZE : String = "SIZE"`
  Property name for file system's total size.
- `<<constant>>AVAILABLE_SPACE : String := "AVAILABLE_SPACE"`
  Property name for file system's available unused space.
- `<<constant>>CREATED_TIME_ID : String = "CREATED_TIME"`
  The CREATED_TIME_ID is the identifier for the created time file property. A created time property indicates the time the file was created.
- `<<constant>>MODIFIED_TIME_ID : String = "MODIFIED_TIME"`
  The MODIFIED_TIME_ID is the identifier for the modified time file property. The modified time property is the time the file data was last modified.
- `<<constant>>LAST_ACCESS_TIME_ID : String = "LAST_ACCESS_TIME"`
  The LAST_ACCESS_TIME_ID is the identifier for the last access time file property. The last access time property is the time the file was last accessed (e.g., read).

Operations

- `copy(in sourceFileName : String, in destinationFileName : String) : {raises = ( InvalidFileName, FileException )}`
  The copy operation provides the ability to copy a regular file (non-directory) to another regular file. The copy operation shall copy the source file with the specified sourceFileName to the destination file with the specified destinationFileName. If the destination file already exists, and the sourceFileName and the destinationFileName are different, the copy operation shall overwrite the destination file. The copy operation shall raise the FileException when a file-related error occurs. The copy operation shall raise the InvalidFileName exception when the source or destination filename is not a valid file name or not an absolute pathname. The copy operation shall raise the InvalidFileName exception when the destination file name is identical to the source file name (i.e., attempting to copy on top of itself).
- `create(in fileName : String, return FileComponent) : {raises = ( InvalidFileName, FileException )}`

The create operation provides the ability to create a new file on the FileSystem. The create operation shall create a new File based upon the fileName attribute. The create operation shall return a File component reference to the opened file. The create operation shall raise the FileException if the file already exists or another file error occurred. The create operation shall raise the InvalidFileName exception when a fileName is not a valid file name or not an absolute pathname or path prefix does not exist.

- `exists(in fileName : String, return Boolean) : {raises = ( InvalidFileName )}`
  The exists operation checks to see if a file exists based on the fileName parameter. The exists operation shall return True if the file exists, or False if it does not. The exists operation shall raise the InvalidFileName exception when fileName is not a valid file name or not an absolute pathname.

- `list(in pattern : String, return FileInformationSequence) : {raises = (FileException, InvalidFileName)}`
  The list operation provides the ability to obtain a list of files along with their information in the FileSystem according to a given search pattern. The list operation can return information for one file or a set of files. The list operation shall return a FileInformationSequence for files that match the wildcard criteria specified in the input pattern parameter. The list operation shall return a zero length sequence when no files are located that match the input pattern parameter. The list operation shall raise the InvalidFileName exception when the input pattern does not start with a slash "/" or cannot be interpreted due to unexpected characters. The list operation shall raise the FileException when a file-related error occurs.

- `open(in filename : String, in read_Only : Boolean, return : FileComponent) : {raises = ( InvalidFileName,  FileException )}`
  The open operation provides the ability to open a file for read or write. The open operation shall open a file based upon the input fileName. The read_Only parameter indicates if the file should be opened for read access only. The open operation shall open the file for read only access when the read_Only parameter is True. The open operation shall return a File component parameter on successful completion. The returned File's filePointer attribute shall be set to the beginning of the file when the read_Only parameter is true, otherwise the File's filePointer attribute is set to the end of the file. If the file is opened with the read_Only flag set to true, then writes to the file will be considered an error. The open operation shall raise the FileException if the file does not exist or another file error occurred. The open operation shall raise the InvalidFileName exception when the filename is not a valid file name or not an absolute pathname.

- `mkdir(in directoryName : String) : {raises( InvalidFileName, FileException )}`
  The mkdir operation provides the ability to create a directory on the file system. The mkdir operation creates a FileSystem directory based on the directoryName given. The mkdir operation shall create all parent directories required to create the directoryName path given. The mkdir operation shall raise the FileException if a file-related error occurred during the operation. The mkdir operation shall raise the InvalidFileName exception when the directoryName is not a valid directory name.

- `query(inout fileSystemProperties : Properties) : {raises( UnknownFileSystemProperties )}`
  The query operation provides the ability to retrieve information about a file system. The query operation shall return file system information to the calling client based upon the given fileSystemProperties' ID. At a minimum, the query operation shall support the following property Ids in the fileSystemProperties parameter:

  - SIZE - an ID value of "SIZE" causes query to return an unsigned long long containing the file system size (in octet(s) <<datatype>>).

  - AVAILABLE SPACE - an ID value of "AVAILABLE SPACE" causes the query operation to return an unsigned long long containing the available space on the file system (in octet(s) <<datatype>>). The query operation shall raise the UnknownFileSystemProperties exception when the given file system property is not recognized.

- `remove(in fileName : String) : {raises = ( FileException, InvalidFileName )}`
  The remove operation provides the ability to remove a regular file (non-directory) from a file system. The remove operation shall remove the file with the given fileName. The remove operation shall raise the InvalidFileName exception when the fileName is not a valid fileName or not an absolute pathname. The remove operation shall raise the FileException when a file-related error occurs.
- `rmdir(in directoryName : String) : {raises( InvalidFileName, FileException )}`
  The rmdir operation provides the ability to remove a directory from the file system. The rmdir operation removes a FileSystem directory, based on the directoryName given, only if the directory is empty (no files exist in directory). The rmdir operation shall raise the FileException when the directory does not exist, if the directory is not empty, or another file-related error occurred. The rmdir operation shall raise the InvalidFileName exception when the directoryName is not a valid directory name.

Types and Exceptions

- `<<enumeration>>FileType (PLAIN, DIRECTORY, FILE_SYSTEM)`
  The FileType indicates the type of file entry. A FILE_SYSTEM can have PLAIN or DIRECTORY files and mounted file systems contained in a FileSystem.
- `FileInformationType ( name : String, kind : FileType, size : ULongLong, fileProperties : Properties)`
  The FileInformationType indicates the information returned for a file. Not all the fields in the FileInformationType are applicable for all file systems. The FileInformationType attributes are:
  - The name indicates the simple name of the file.
  - The kind indicates the type of the file entry.
  - The size indicates the size in Octet(s).
  - The FileProperties indicates other optional File properties that could be given out.
- `FileInformationSequence.`
  The FileInformationSequence type defines an unbounded sequence of FileInformationTypes

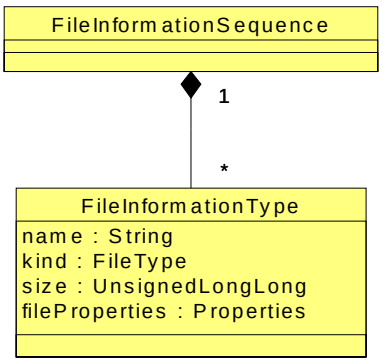

Figure 7.26 - FileInformationSequence

- `<<exception>>UnknownFileSystemProperties`
  The UnknownFileSystemProperties exception, specialization of UnknownProperties, indicates a set of properties unknown by the component.

Constraints

Valid characters for a file name and file absolute pathname shall adhere to POSIX compliant file naming conventions. At a minimum, a regular file name length of 40 characters shall be supported. At a minimum, a combined path prefix and ending regular file name length of 1024 characters shall be supported.

At a minimum, the FileSystem shall support the FileInformationType attributes: name, kind, and size information for a file. Examples of other file properties that may be specified for fileProperties are created time, modified time, and last access time as stated in Types and Exceptions above. The value for these properties shall be unsigned long long and measured in seconds since 00:00:00 UTC, Jan.1, 1970.

7.2.1.3.2 File Services Stereotypes

The File Services components are identified in Table 7.7. File services stereotypes and their relationship are graphically depicted in Figure 7.24.

Table 7.7 - File Services Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
FileComponent	Component	N/A		See constraints in section below	Reads and writes a file within a FileSystem.
FileManagerComponent	Component	N/A		See constraints in section below	Provides a single interface to multiple, distributed FileSystems
FileSystemComponent	Component	N/A		See constraints in section below	Remotely accesses a physical file system.

7.2.1.3.2.1 FileComponent

Description

The FileComponent, as shown in Figure 7.24, provides the ability to read and write files residing within a potentially distributed FileSystemComponent. A file can be thought of conceptually as a sequence of Octet(s) with a current filePointer describing where the next read or write will occur. This filePointer points to the beginning of the file upon construction of the file object.

Constraints

A FileComponent shall realize the File interface.

7.2.1.3.2.2 FileManagerComponent

Description

The FileManagerComponent, as shown in Figure 7.24, realizes the FileManager interface and extends that interface by adding mount and unmount operations. This allows multiple, distributed FileSystemComponents to be accessed through a FileManagerComponent. The FileManagerComponent appears as a single file system although the actual file storage may span multiple physical file systems. This is called a federated file system. A federated file system is created using the mount and unmount operations.

M1 Associations

- `mountedFileSystem:FileSystemComponent[*]`
  MountedFileSystem(s) that are associated with a FileManagerComponent.

Constraints

The FileManagerComponent shall realize the FileManager interface.

7.2.1.3.2.3 FileSystemComponent

Description

A FileSystemComponent, as shown in Figure 7.24, realizes the FileSystem interface and may be associated with a File-Manager and consist of many FileComponents.

Associations

- `fileManager : FileManagerComponent[*]`
  A file system can be associated with many FileManagerComponents.

Constraints

The FileSystemComponent shall realize the FileSystem interface.

7.2.2 Platform Management

This section defines the stereotypes for platform management. Platform management involves the management of the domain, inclusive of its devices and services. The platform management stereotypes are categorized by domain and device management. The details of these categories are described in the following Domain Management and Device Management subsections.

7.2.2.1 Domain Management

7.2.2.1.1 Domain Management Interfaces

This section defines the interfaces for Domain management. The types of capabilities offered by Domain management are categorized as follows:

1. Domain Registration Management - provides the mechanism for registering and unregistering DeviceManager’s services within a Domain.
2. Domain Installation Management - provides the mechanism for installing and nonstaining applications within a Domain.
3. Domain Manager - provides the mechanism for retrieving a domain’s components.
4. Domain Event Channels- provides the mechanism for managing Domain's event channels connections.

Types and Exceptions

- `<<exception>>InvalidProfile`
  The InvalidProfile exception indicates an invalid profile error. A profile error indicates an invalid Descriptor (e.g., component description, Application description, etc.).

7.2.2.1.1.1 DomainEventChannels

Description

The DomainEventChannels interface, as shown in an association in Figure 7.28, provides domain event channel registration capabilities. The interface provides the capabilities of adding and removing connections to event channels in a domain.

Operations

- `registerWithEventChannel(in registeringObject: Object, in registeringId: String, in eventChannelName : String): {raises = (InvalidObjectReference, InvalidEventChannelName, AlreadyConnected)}`
  The registerWithEventChannel operation shall connect a consumer (registeringObject input parameter) to a domain's event channel as indicated by the input eventChannelName parameter.

  The registerWithEventChannel operation shall raise the InvalidObjectReference exception when the input registeringObject parameter contains an invalid reference to an event channel consumer type interface. The registerWithEventChannel operation shall raise the InvalidEventChannelName exception when the input eventChannelName parameter contains an invalid event channel name. The registerWithEventChannel operation shall raise the AlreadyConnected exception when the input eventChannelName parameter references an event channel that is connected to the consumer identified by the input registeringId parameter.

- `unregisterFromEventChannel(in unregisteringId: String, in eventChannelName: String): { raises = (InvalidEventChannelName, NotConnected)}`
  The unregisterFromEventChannel operation shall disconnect a consumer (unregisteringId input parameter) from a domain's event channel as indicated by the eventChannelName parameter.

  The unregisterFromEventChannel operation shall raise the InvalidEventChannelName exception when the input eventChannelName parameter contains an invalid reference to an event channel. The unregisterFromEventChannel operation shall raise the NotConnected exception when the input parameter unregisteringId parameter is not connected to the specified input event channel.

Types and Exceptions

- `<<exception>>AlreadyConnected`
  The AlreadyConnected exception indicates that a registering consumer is already connected to the specified event channel.

- `<<exception>>InvalidEventChannelName`
  The InvalidEventChannelName exception indicates that the event channel with that name does not exist within the domain.

- `<<exception>> NotConnected`
  The NotConnected exception indicates that the unregistering consumer was not connected to the specified event channel.

7.2.2.1.1.2 DomainInstallation

Description

The DomainInstallation interface, as shown in an association in Figure 7.28, defines domain application installation capabilities. The interface provides the capabilities of adding and removing applications from a domain.

Operations

- `installApplication(in profileFileName: String): {raises = ( InvalidProfile, InvalidFileName, ApplicationInstallationError, ApplicationAlreadyInstalled)}`
  The installApplication operation is used to install new application artifacts and descriptors in the domain. A SWRadioDeployment::Application Deployment::ApplicationFactory is created in the domain as a result of successful installation. The profileFileName is the absolute path of the profile filename. The installApplication operation shall verify that the Application's descriptor file exists in the domain and that all the files the Application depends on are also resident.

  The installApplication operation shall raise the ApplicationInstallationError exception when the installation of the Application file(s) is not successfully completed. The installApplication operation shall raise the ApplicationInstallationError exception when the to-be-installed application's identifier (specified in the application's descriptor referenced by the profileFileName input parameter) is the same as a previously registered application. The installApplication operation shall raise the InvalidFileName exception when the input file or any referenced file name does not exist in the file system as defined in the absolute path of the input profileFileName. The installApplication operation shall raise the InvalidProfile exception when the input file or any referenced descriptor file is not compliant with its descriptor definition.

  The installApplication operation shall raise the ApplicationAlreadyInstalled exception when the application descriptor element id attribute of the referenced application (as denoted by the input profileFileName) is the same as a previously registered application.

- `uninstallApplication(in applicationId: String): {raises = (InvalidIdentifier, ApplicationUninstallationError)}`
  The uninstallApplication operation is used to uninstall an application from the domain. The applicationId corresponds to the identifier in the SWRadioDeployment::Application Deployment::ApplicationFactory. The uninstallApplication operation shall remove the ApplicationFactory from the domain.

  The uninstallApplication operation shall raise the InvalidIdentifier exception when the applicationId is invalid. The uninstallApplication operation shall raise the ApplicationUninstallationError exception when an internal error causes an unsuccessful uninstallation of the Application.

Types and Exceptions

- `<<exception>>ApplicationAlreadyInstalled`
  The ApplicationAlreadyInstalled exception indicates that the application being installed is already installed.

- `<<exception>>ApplicationInstallationError ( ErrorNumberType: errorNumber, msg: String )`
  The ApplicationInstallationError exception, a type of System Exception, is raised when an application installation has not completed correctly. The error number indicates the type of error (e.g., CF_EINVAL, CF_ENAMETOOLONG, CF_ENOENT, CF_ENOMEM, CF_ENOSPC, CF_ENOTDIR, CF_ENXIO). The message is component-dependent, providing additional information describing the reason for the error.

- `<<exception>>ApplicationUninstallationError`
  The ApplicationUninstallationError exception, a type of SystemException, is raised when an application uninstallation has not completed correctly.

- `<<exception>> InvalidIdentifier`
  The InvalidIdentifier exception indicates an application identifier is invalid.

Semantics

An installer service typically invokes these operations when adding or removing an ApplicationFactory (installed Application) after creating or prior to deleting the associated files.

7.2.2.1.1.3 DeviceManagerRegistration

Description

The DeviceManagerRegistration interface, as shown in an association in Figure 7.28, defines domain service registration capabilities. The interface provides the capabilities of adding and removing DeviceManager's services from a domain.

Operations

- `registerDeviceManager(in deviceMgr: DeviceManagerComponent): {raises = (InvalidObjectReference, InvalidProfile, RegisterError )}`
  The registerDeviceManager operation shall add the input deviceMgr to the domain, if it does not already exist. The registerDeviceManager operation shall add each deviceMgr's ServiceComponent attributes (e.g., identifier, name/label, characteristic and capacity properties, etc.) to the domain as specified in the deviceMgr's registeredServices attribute. The registerDeviceManager operation shall establish any connections specified in the DeviceManager's descriptor that are possible with the current set of domain registered ServiceComponent(s)--others are left unconnected pending future ServiceComponent registrations. The registerDeviceManager operation shall establish any pending connections from previously registered DeviceManagers if any of the newly registered ServiceComponent(s) complete these connections.

  For connections established for an EventService's EventChannel, the registerDeviceManager operation shall connect to the event channel as specified in the deviceMgr's descriptor. If the EventChannel does not exist, the registerDeviceManager operation shall create the EventChannel.

  The registerDeviceManager operation may obtain all the Descriptor(s) from the registering DeviceManager's FileSystem.

  The registerDeviceManager operation shall mount the deviceMgr's FileSystem to the domain. The mounted FileSystem name shall have the format, "/DomainName/HostName," where DomainName is the name of the domain and HostName is the input deviceMgr's label attribute.

  The registerDeviceManager operation shall raise the InvalidObjectReference exception when the input parameter deviceMgr contains an invalid reference to a DeviceManager interface.

  The registerDeviceManager operation shall raise the RegisterError exception when an internal error exists which causes an unsuccessful registration.

- `unregisterDeviceManager(in deviceMgr: DeviceManagerComponent): {raises = (InvalidObjectReference, UnregisterError) }`
  The unregisterDeviceManager operation is used to unregister a DeviceManager component from domain. The unregisterDeviceManager operation shall unregister input deviceMgr component and its ServiceComponent(s) from the domain. The unregisterDeviceManager operation shall disconnect the established connections (including those made to an EventService's Event Channel) involving the unregistering DeviceManager. Any such broken connections to components remaining from other DeviceManager's DCD files shall be considered as "pending" future connections. The unregisterDeviceManager operation may destroy the EventService EventChannel when no more consumers and producers are connected to it.

The unregisterDeviceManager operation shall unmount the deviceMgr's FileSystem from the domain.

The unregisterDeviceManager operation shall raise the InvalidObjectReference when the input parameter deviceMgr contains an invalid reference to a DeviceManager interface. The unregisterDeviceManager operation shall raise the UnregisterError exception when an internal error exists that causes an unsuccessful unregistration.

- `registerService(in registeringService: ServiceComponent, in registeredDeviceMgr: DeviceManagerComponent, in name: String): {raises = (InvalidObjectReference, DeviceManagerNotRegistered, RegisterError) }`
  The registerService operation registers a DeviceManager's ServiceComponent to a domain.
  The registerService operation shall add the registeringService and the registeringService's attributes (e.g., identifier, name/label, descriptor's characteristic and capacity properties, etc.) to the domain, if the registeringService does not already exist and registeredDeviceManager exists in the domain.

  The registerService operation shall establish any required port connections pending from any previously registered DeviceManagers that are satisfied by the newly registered ServiceComponent, using the PortConnector and PortSupplier interfaces.

  The registerService operation shall raise the InvalidProfile exception when registeringService's descriptor is invalid. The registerService operation shall raise a DeviceManagerNotRegistered exception when the input registeredDeviceMger is not registered with the domain. The registerService operation shall raise the InvalidObjectReference exception when input parameters registeringService or registeredServiceMgr contain an invalid reference. The registerService operation shall raise the RegisterError exception when an internal error exists that causes an unsuccessful registration.

- `unregisterService(in unregisteringService: ServiceComponent, in name: String): {raises = ( InvalidObjectReference, UnregisterError)}`
  The unregisterService operation shall remove a ServiceComponent entry from the domain.
  The unregisterService operation shall disconnect the unregisteringService's port connections. Such connections to the remaining components shall then be considered as "pending" future connections. The unregisterService operation may destroy the EventService's EventChannel when no more consumers and producers are connected to it.

  The unregisterService operation shall raise the InvalidObjectReference exception when the input parameter contains an invalid reference to a ServiceComponent interface. The unregisterService operation shall raise the UnregisterError exception when an internal error exists which causes an unsuccessful unregistration.

Types and Exceptions

- `<<exception>>DeviceManagerNotRegistered`
  The DeviceManagerNotRegistered exception indicates the registering Service's DeviceManager is not registered in the domain. A Service's DeviceManager has to be registered prior to a service registration to the domain.

- `<<exception>>RegisterError`
  The RegisterError exception, a type of System Exception, indicates that an internal error has occurred to prevent domain registration operations from successful completion.

- `<<exception>>UnregisterError`
  The UnregisterError exception, a type of SystemException, indicates that an internal error has occurred to prevent domain unregistration operations from successful completion.

Semantics

The DeviceManagerRegistration interface provides the mechanisms for components, such as DeviceManagers, to register their ServiceComponent(s) (e.g., ManagedServiceComponent or DeviceComponent) for a specific domain. As ServiceComponent(s) are removed from the environment, the interface provides the capability of removing them from the domain. Setting up connections for a registeringService is usually done for DeviceComponents that need an EventChannel, LogService, etc. As ServiceComponent(s) are made available to a domain, they become available for the domain's Application(s) deployment usage.

The behavior for duplicated registering DeviceManager or ServiceComponent may result in a RegisterError exception being thrown or the duplicated registration may be ignored. The duplicated registration behavior is left up to PSMs, PIMs, or profiles that extend the profile.

7.2.2.1.1.4 DomainManager

Description

The DomainManager interface as shown in Figure 7.27 describes the definition and relationships that are common for all domain managers. The DomainManager provides the capabilities of retrieving a domain's components and profile.

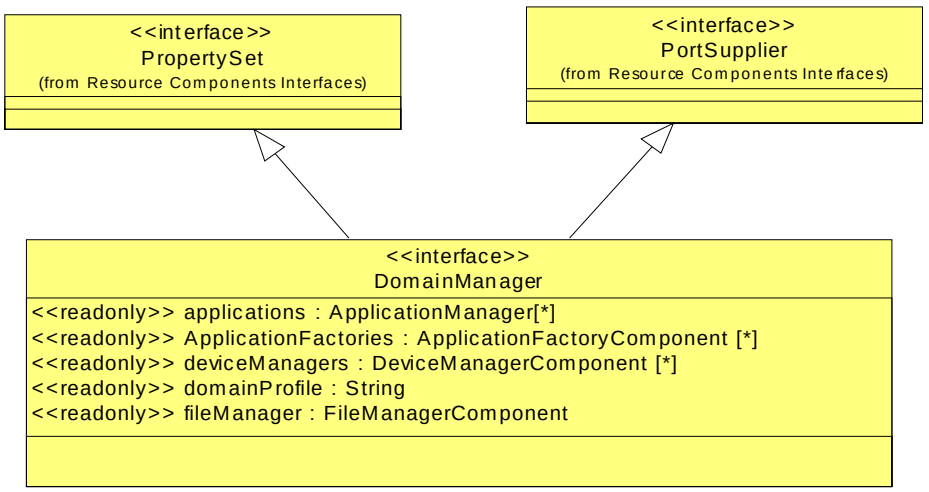

Figure 7.27 - Domain Manager Definition

Attributes

- `<<readonly>>applications : ApplicationManager [*]`
  The readonly applications attribute contains a sequence of instantiated applications in the domain. The applications attribute shall contain the list of Deployment::Application Deployment::Application Deployment Stereotypes::ApplicationManager(s) that have been instantiated (e.g., created by ApplicationFactory) within the domain.

- `<<readonly>>applicationFactories : ApplicationFactoryComponent [*]`
  The readonly applicationFactories attribute contains a list of ApplicationFactories in the domain. The applicationFactories attribute shall contain a list of one Deployment::Application Deployment::Application Deployment Stereotypes::ApplicationFactory per Application successfully installed (i.e., no exception raised by the DomainInstallation::install).

- `<<readonly>>deviceManagers : DeviceManagerComponent [*]`
  The readonly deviceManagers attribute contains a sequence of DeviceManagerComponents in the domain. The deviceManagers attribute shall contain a list of registered DeviceManagerComponents that have registered with the DomainManagerComponent.

- `<<readonly>>domainManagerProfile : String`
  The readonly domainManagerProfile attribute contains a file reference to the domain configuration descriptor. The descriptor provides configuration information for a domain. Files referenced within the descriptor will have to be obtained from the domain's fileMgr attribute.

- `<<readonly>>fileMgr : FileManagerComponent`
  The readonly fileMgr attribute contains the DomainManager's RadioServices::File Services::File Services Stereotypes::FileManagerComponent.

7.2.2.1.2 Domain Management Stereotypes

This section defines the stereotypes for Domain management. The Domain management stereotypes are depicted in Table 7.8, which are extensions of the UML 2.0 Component (UML2.0::Components::BasicComponents).

Table 7.8 - DomainManagement Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
DomainManager Component	Component	Component		See constraints in section below	Provides Domain Retrieval capability and represents a component that manages a domain.

7.2.2.1.2.1 DomainManagerComponent

Description

The DomainManagerComponent as shown in Table 7.8 and Figure 7.28 describes the definition and relationships that are common for all domain managers. The DomainManagerComponent provides the capabilities of retrieving a domain's components and profile.

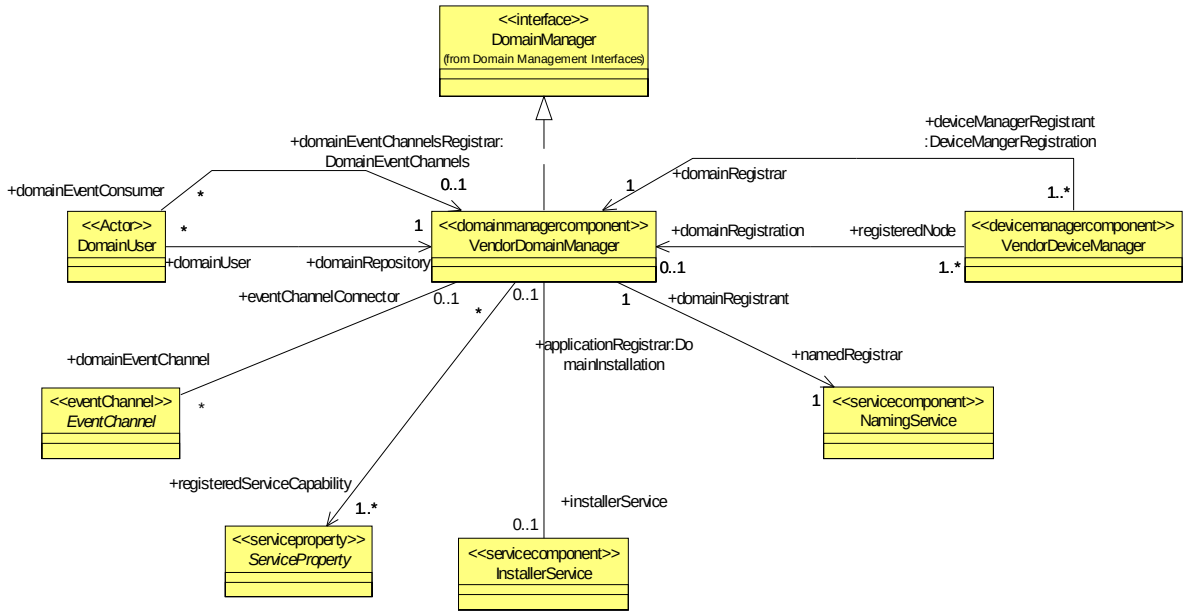

Figure 7.28 - DomainManagerComponent M1 Illustration

M1 Associations

- `domainEventChannel: EventChannel [*]`
  A DomainManagerComponent may be associated with many event channels. Refer to Domain Event Channels section below for description of domain event channels and event types.

- `namedRegistrar: NamingService [1]`
  The NamingService that contains a named DomainManagerComponent reference.

- `registeredServiceCapability: ServiceProperty [1..*]`
  The registedServiceCapability(s) are the set of ServiceComponent's ServiceProperty(s) registered and/or known by the DomainManagerComponent.

Constraints

The identifier attribute shall contain a unique identifier for a DomainManagerComponent. The identifier shall be identical to the id attribute of the DomainManagerComponent's configuration descriptor as specified by the domainProfile attribute.

During component construction the DomainManagerComponent shall register itself with the NamingService. During NamingService registration the DomainManagerComponent shall create a "naming context" using "/DomainName" as its name.ID component and "" (Null string) as its name.kind component, then create a "name binding" to the "/DomainName" naming context using "/DomainManager" as its name.ID component, "" (Null string) as its name.kind component, and the DomainManager's object reference.

The DomainManagerComponent shall create its own FileManager component that consists of all registered DeviceManager's FileSystems.

The DomainManagerComponent shall restore ApplicationFactories after startup for Application(s) that were previously installed by the DomainManagerComponent installApplication operation. The DomainManagerComponent shall add the restored ApplicationFactories to the DomainManagerComponent's applicationFactories attribute.

If the DomainManagerComponent has the association role of a domainEventChannelsRegistrar, then the DomainManagerComponent shall provide a port that provides the IDomainEventChannels service and the port name is DomainEventChannelsPort, and support the DomainOutgoingEventChannel and DomainIncomingEventChannel.

If the DomainManagerComponent has the association role of a domainRegistration, then the DomainManagerComponent shall provide a port that provides the DeviceManagerRegistration service and the port name is DomainDeviceMgrRegPort.

If the DomainManagerComponent has the association role of an applicationRegistrar, then the DomainManagerComponent shall provide a port that provides the ApplicationInstallation service and the port name is DomainInstallationPort.

If the DomainManagerComponent has a LogService required port, then the DomainManagerComponent shall provide a port that provides LogService admin service and the port name is DomainLogAdminPort.

If the DomainManagerComponent has a LogService required port, then the DomainManagerComponent shall provide a port that provides LogService consumer service and the port name is DomainLogConsumerPort.

Each registered ServiceProperty with the same identification shall be the same type within DomainManagerComponent.

Semantics

The set of interfaces realized by a DomainManagerComponent depends on the system the DomainManagerComponent is built for. As shown in Figure 7.28, the DomainManagerComponent could realize PortSupplier and PropertySet interfaces and inherits from Component. The types of ServiceArtifactProperty(s) supported by a DomainManagerComponent are implementation specific. A DomainManagerComponent implementation may constrain the types of ServiceArtifactProperty(s) which would be reflected in its registration behavior.

The DomainManagerComponent may upon successful add of a component to its applications, applicationFactories, or deviceManagers attribute or registered Service send an event to the Outgoing Domain ManagementEventChannel with event data consisting of a DomainManagementObjectAddedEventType. The DomainManagementObjectAddedEventType event data shall be populated as follows when issued:

1. The producerId is the identifier attribute of the DomainManagerComponent.
2. The sourceId is the identifier attribute of the created (ApplicationManager), installed (ApplicationFactory), or registered (DeviceManager or Service) component to the domain.
3. The sourceName is the name attribute of the added component to the domain.
4. The sourceHandle is the component reference added to the domain.
5. The sourceCategory is the type of component added to the domain.

The DomainManagerComponent may upon successful removal of a component from its applications, applicationFactories, or deviceManagers attribute or unregistered Service send an event to the Outgoing Domain ManagementEventChannel with event data consisting of a DomainManagementObjectRemovedEventType. The DomainManagementObjectRemovedEventType event data shall be populated as follows when issued:

1. The producerId is the identifier attribute of the DomainManagerComponent.

2. The sourceId is the identifier attribute of the component uninstalled (ApplicationFactory), released (ApplicationManager), or unregistered (DeviceManager or Service).

3. The sourceName is the name attribute of the removed component from the domain.

4. The sourceCategory is the type of component removed from the domain.

The DomainManagerComponent shall produce DomainManagementObjectRemovedEventType and DomainManagementObjectAddedEventType when the DomainManagerComponent has the association role of a domainEventChannelsRegistrar.

7.2.2.2 Device Management

This section defines the stereotypes and interfaces for device management, which are described in the two subsections: 7.2.2.2.1 and .7.2.2.2.2.

7.2.2.2.1 Device Management Interfaces

This section defines the interfaces for device management. The types of capabilities offered by device management are categorized as follows:

1. Service Registration Management - provides the mechanism for registering and unregistering services within a device.

2. DeviceManager - provides the mechanism for retrieving node's services and information.

7.2.2.2.1.1 DeviceManager

Description

The DeviceManager interface as shown in Figure 7.29 defines the attributes and operations relationships that are common for all node managers. A DeviceManager interface is used to manage the ServiceComponent(s) on a node and to retrieve information about a node or device manager.

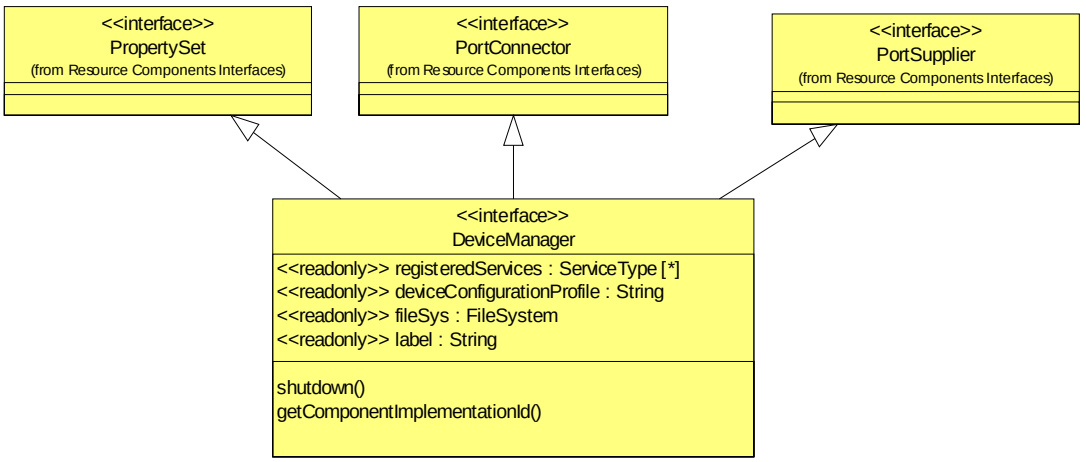

Figure 7.29 - DeviceManager Definition

Attributes

- `<<readonly>>label: String`
  The readonly label attribute contains a node's meaningful name.
- `<<readonly>>fileSys: FileSystemComponent`
  The readonly fileSys attribute contains the FileSystem associated with this node or a nil component reference if no FileSystem is associated with this node.
- `<<readonly>>deviceConfigurationProfile : String`
  The readonly deviceConfigurationProfile attribute contains information on the initial configuration for the node. Files referenced within the profile are obtained using the fileSys attribute.
- `<<readonly>>registeredServices : ServiceSequence`
  The readonly registeredServices attribute contains a list of Services that have registered with a node or a sequence length of zero if no Services have registered with the node.

Operations

- `getComponentImplementationId (in componentInstantiationId: String, return String):`
  The getComponentImplementationId operation shall return the implementation ID used to create the component identified by the input componentInstantiationId parameter. The implementation ID corresponds to the ServiceExecutableCode's implementation id in a component implementation descriptor that was used to manifest a ServiceComponent. The getComponentImplementationId operation shall return an empty string when the input componentInstantiationId parameter does not match a ServiceExecutableCode's implementation ID deployed by the DeviceManager. This operation does not raise any exceptions.
- `shutdown ()`
  The shutdown operation provides the mechanism to terminate a DeviceManager. The shutdown operation shall unregister the DeviceManager from the DomainManagerComponent.

  The shutdown operation shall release all of the DeviceManager's registered ServiceComponent(s) (DeviceManager's registeredServices attribute) that support the Ilifecycle interface and started up by the DeviceManager.

  The shutdown operation shall terminate the execution of each ExecutableCode that was created as specified in the deviceConfigurationProfile attribute. For a released (releaseObject operation) ServiceComponent, the termination shall take place after the ServiceComponent has unregistered with the DeviceManager.

  The shutdown operation shall remove the DeviceManager from the environment when all of the released registered Services are unregistered from the DeviceManager. This operation does not return any value and does not raise any exceptions.

Types and Exceptions

- `ServiceSequence`
  The ServiceSequence type defines an unbounded sequence of ServiceTypes.
- `ServiceType( serviceObject: ServiceComponent, serviceName: String)`
  This structure provides the Service reference and name of Service that have registered with the node.

7.2.2.2.1.2 ServiceRegistration

Description

The ServiceRegistration interface defines node service registration capabilities. The interface provides the capabilities of adding and removing ServiceComponent(s) from a node.

Operations

- `registerService(in registeringService: ServiceComponent, in name: String): {raises = (InvalidObjectReference) }`
  The registerService operation provides the mechanism to register a ServiceComponent with a node. The registeringService is ignored when duplicated. The registerService operation shall raise the InvalidObjectReference exception when the input registeringService is a nil component reference.
- `unregisterService(in unregisteringService: ServiceComponent, in name: String): {raises = ( InvalidObjectReference)}`
  The unregisterService operation provides the mechanism to unregister a ServiceComponent from a node. The unregisterService operation shall raise the InvalidObjectReference when the input registeredService is a nil component reference or does not exist in the node.

Semantics

The ServiceRegistration interface provides the mechanisms for ServiceComponent(s) started up on a node to register to a node or device manager that is managing a node. As ServiceComponent(s) are removed from node environment, the interface provides the capability of removing them from a node or device manager. ServiceComponents managed by a node manager can also include DeviceComponent(s).

7.2.2.2.2 Device Management Stereotypes

This section defines the stereotypes for device management. The device management stereotypes are depicted in Table 7.9, which are extensions of UML 2.0 Component (UML2.0::Components::BasicComponents). The details of each classifier are described in the following subsections.

Table 7.9 - Node Management Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
DeviceManagerComponent	Component	Component		See constraints in section below	Manages a node and its services.

7.2.2.2.2.1 DeviceManagerComponent

Description

The DeviceManagerComponent a type of Component as shown in Table 7.9. Figure 7.30 denotes the relationships that are common for all node managers. A DeviceManagerComponent manages the ServiceComponent(s) on a node.

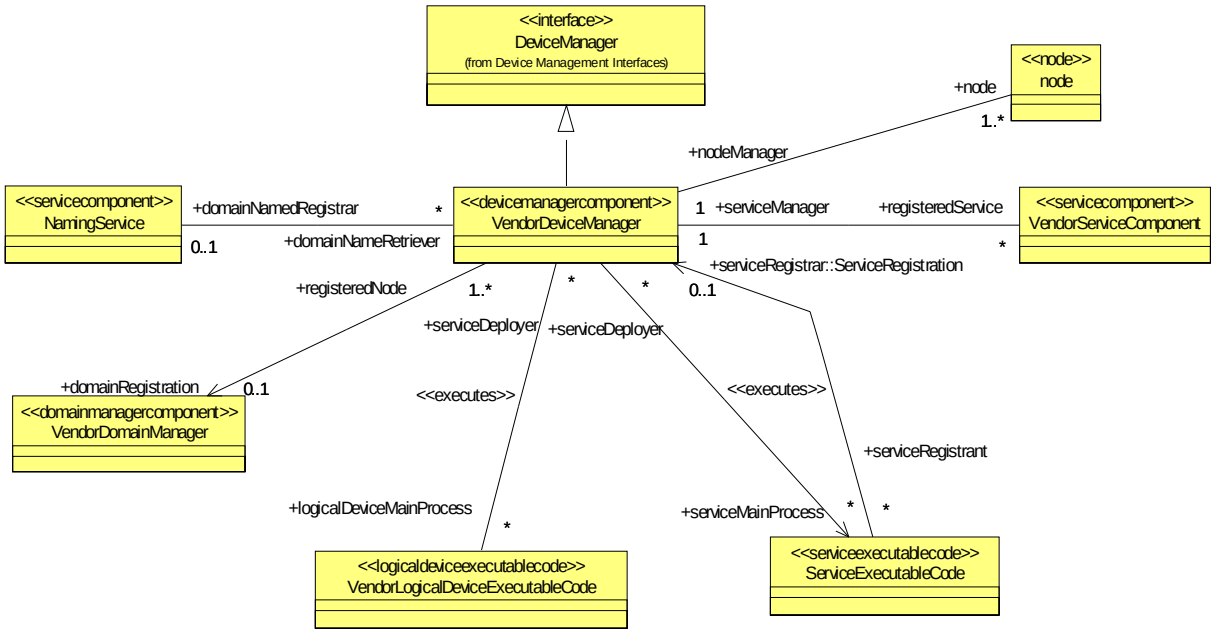

Figure 7.30 - DeviceManager M1 Illustration

M1 Associations

- `node: Node[1..*]`
  One to many nodes can be managed by a DeviceManagerComponent.
- `domainNamedRegistrar: NamingService [0..1]`
  A NamingService contains the named DomainManagerComponent's component reference that is obtained by a DeviceManager that needs to register to a DomainManagerComponent.
- `domainNodeRegistrar: DomainManagerComponent [0..1]`
  A DomainManagerComponent harnesses all nodes' services and capabilities within the domain.
- `logicalDeviceMainProcess: LogicalDeviceExecutableCode [*]`
  A logical device main process that manifests a DeviceComponent can be executed by a DeviceManagerComponent.
- `registeredService: ServiceComponent[1..*]`
  A set of ServiceComponent(s) (e.g., DeviceComponents, etc.) registered to a DeviceManagerComponent.
- `serviceMainProcess: ServiceExecutableCode [*]`
  A service component's main process that manifests a ServiceComponent can be executed by a DeviceManagerComponent.

Constraints

The registeredServices attribute shall contain a list of registered Services::ServiceComponent(s) that have registered with the DeviceManagerComponent. Each registeredservice in the registeredServices attribute shall be registered with the DeviceManagerComponent's associated DomainManagerComponent when the DeviceManagerComponent registers with the DomainManagerComponent.

Each unregistered ServiceComponent shall be unregistered from the DeviceManagerComponent's associated DomainManagerComponent when the unregistered ServiceComponent is registered with the DeviceManagerComponent and the DeviceManagerComponent is not shutting down. If a Deployment:: Artifacts::ServiceExecutableCode was started up by the DeviceManagerComponent as specified in the deviceConfiguration attribute, then the DeviceManagerComponent shall terminate the ServiceExecutableCode and deallocate capacity to the device the ServiceExecutableCode was deployed on.

If the DeviceManagerComponent has the association role of a serviceRegistrar, then the DeviceManagerComponent shall provide a port that provides the ServiceRegistration service and the port name is ServiceRegistration.

The DeviceManagerComponent shall realize the DeviceManager interface.

Semantics

The DeviceManagerComponent provides the capability of starting up ServiceComponent(s)' main processes on a given node by the deviceConfigurationDescriptor attribute. ServiceComponent(s) started up also include DeviceComponent(s) that are a type of ManagedServiceComponent. A DeviceManagerComponent registers to a DomainManagerComponent using the deviceConfigurationProfile descriptor information.

The DeviceManagerComponent upon start up shall register itself with a DomainManagerComponent as specified in the deviceConfigurationProfile attribute. A DeviceManagerComponent shall use its deviceConfigurationProfile attribute for determining:

1. ServiceComponent(s) to be deployed for this DeviceManagerComponent (for example, LogService, DeviceComponent),
2. DeviceComponents to be created for this DeviceManagerComponent,
3. ServiceComponent(s) to be deployed on (executing on) another Device,
4. DeviceComponents to be part of another DeviceComponent's composition definition,
5. Mount point names for File Systems,
6. The DeviceManagerComponent's identifier attribute value, and
7. The DeviceManagerComponent's label attribute value.

The DeviceManagerComponent shall create file system components implementing the FileSystem interface. If multiple FileSystems are to be created, the DeviceManagerComponent shall mount created FileSystemComponents to a FileManagerComponent (widened to a FileSystem through the FileSys attribute). Each mounted FileSystemComponent name shall be unique within the DeviceManagerComponent.

If the DeviceManagerComponent deploys a ServiceComponent, the DeviceManagerComponent shall supply execute operation parameters as stated for Deployment:: Artifacts::ServiceExecutableCode. The ServiceComponent Name executable parameter shall be ServiceComponent's usagename element as specified in the deviceManager's DCD. The DeviceManagerComponent shall use a Deployment:: Artifacts::ExecutableCode's user-defined executable parameters as

specified in the component's implementation descriptor. The DeviceManagerComponent shall use an ExecutableCode's stacksize, priority, runtime executable options when specified in the component's implementation descriptor. The DeviceManagerComponent shall allocate the ExecutableCode's capacity requirements against the device the ExecutableCode is deployed on.

If the DeviceManagerComponent deploys a DeviceComponent, the DeviceManagerComponent shall supply execute operation parameters as stated for Deployment:: Artifacts::LogicalDeviceExecutableCode. The following LogicalDeviceExecutableCode's execute operation parameters values shall be:

- The Device Identifier executable parameter shall be ServiceComponent's id element as specified in the DeviceManager's DCD.
- The Profile Name executable parameter shall be ServiceComponent's component implementation descriptor as specified in the DeviceManagerComponent's DCD. The file name shall be the DeviceManagerComponent's full mounted file system file path name.
- Composite Device Component Reference executable parameter shall be a registered DeviceComponent's compositeDevice attribute that corresponds to the composite part relationship as specified in the DeviceManagerComponent's DCD. This parameter is only used when the composite part relationship is specified in the DeviceManagerComponent's DCD.

The DeviceManagerComponent shall initialize and configure ServiceComponents that are started by the DeviceManagerComponent after they have registered with the DeviceManagerComponent provided they realize the Lifecycle and PropertySet interfaces. The DeviceManagerComponent shall configure a registered ServiceComponent, provided the registered ServiceComponent has ConfigureProperty(s).

A ServiceComponent's configuration property values shall only come from the deviceConfigurationProfile descriptor, not from the component's implementation or component definition descriptors.

7.2.2.3 Domain Event Channels

For domain management, a domain may support a number of event channels. Two event channels that a domain may support are:

- IncomingDomainEventChannel - This event channel receives events from the domain's registered components.
- OutgoingDomainEventChannel - This event channel sends events from the domain out to registered domain event consumers.

The types of domain events that could be issued are depicted in the Figure 7.31 and described in the Types and Exceptions section below.

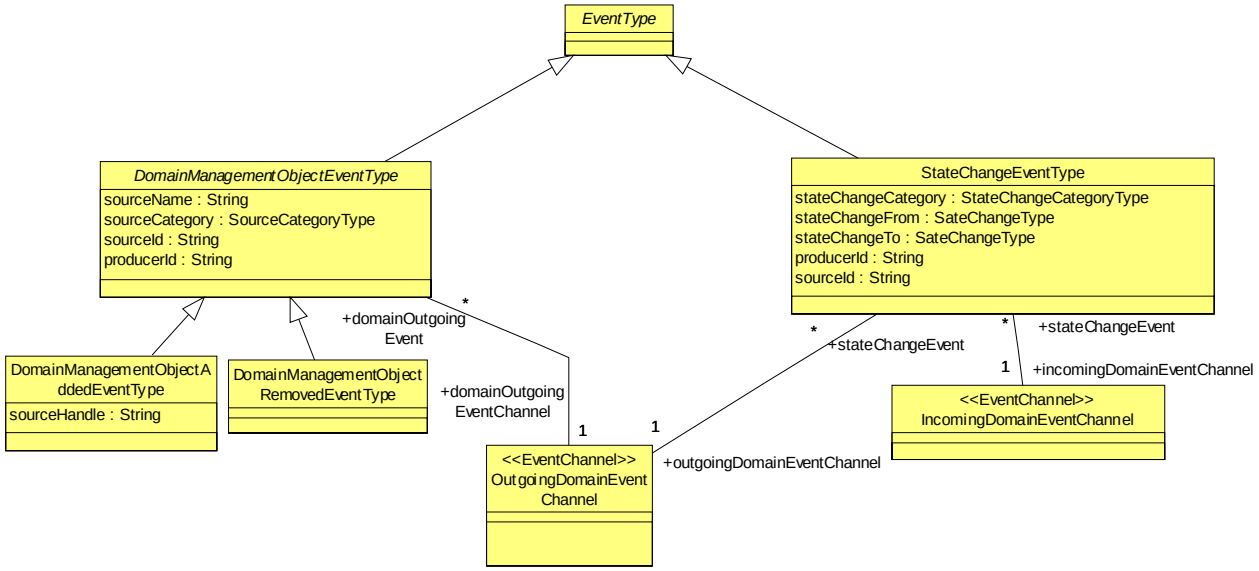

Figure 7.31 - Domain Events Overview

Types and Exceptions

- `DomainManagementObjectAddedEventType( sourceHandle: String)`
  The DomainManagementObjectAddedEventType is a specialization of DomainManagementObjectEventType. This event type indicates a component has been added to the domain. The sourceHandle attribute indicates the component reference of the component added to the domain.
- `DomainManagementObjectEventType ( sourceName: String, sourceCategory: SourceCategoryType, sourceId: String, producerId: String)`
  The DomainManagementObjectRemovedEventType is a specialization of event, which contains information about the event that occurred in the domain.
  - SourceName attribute is the name of source that caused the event.
  - SourceCategory attribute indicates the type of component that caused the event.
  - SourceID is the identifier of the source that caused the event.
  - ProducerId is the identifier of the producer of the event.
- `DomainManagementObjectRemovedEventType`
  The DomainManagementObjectRemovedEventType is a specialization of DomainManagementObjectEventType. This event type indicates a component has been removed from the domain.
- `<<enumeration>>SourceCategoryType( COMM_CHANNEL, DEVICE_MANAGER, DEVICE, DOMAIN_MANAGER,`
  `APPLICATION, APPLICATION_FACTORY, SERVICE))`
  The SourceCategoryType defines the types of components within the domain that can be added or removed. A ManagedService in the domain can also have its state changed.

- `<<enumeration>>StateChangeCategoryType( Administrative_Event_State, Operational_Event_State, Usage_Event_State)`
  The StateChangeCategoryType indicates either an admin, operational, or usage state change.
- `<<enumeration>>StateChangeCategoryType(Administrative_Event_State, Operational_Event_State, Usage_Event_State)`
  The StateChangeCategoryType indicates either an admin, operational, or usage state change.
- `StateChangeEventType( stateChangeCategory:StateChangeCategoryType, stateChangeFrom:StateChangeType, stateChangeTo: StateChangeType, producerId: String, sourceId: String)`
  The StateChangeEventType indicates either an admin, operational, or usage state change. The stateChangeCategory attribute indicates the type of state change. The sourceId attribute indicates the source component's state change and the producerId indicates the component that produced the event.
- `<<enumeration>>StateChangeType( LOCKED, UNLOCKED, SHUTTING_DOWN, ENABLED, DISABLED, IDLE, BUSY, ACTIVE)`
  The StateChangeType indicates the values for state changes. LOCKED, UNLOCKED, and SHUTTING_DOWN values are associated with admin state changes. ENABLED and DISABLED values are associated with operational state changes. IDLE, BUSY, and ACTIVE values are associated with usage state changes.

7.2.3 Deployment

Deployment describes the executable artifacts that are involved in the deployment of applications (e.g., applications), logical devices, and services within an environment. This section also describes the components that are involved in the deployment of and management of Applications in the Application Deployment subsection.

7.2.3.1 Artifacts

This section defines the types and stereotypes for artifacts that are described in the following two subsections respectively, 7.2.3.1.1 and 7.2.3.1.2.

7.2.3.1.1 Artifacts Types

This section defines the types for artifacts that are depicted in Figure 7.5. These types are used to specify a deployment requirement on a device within a platform.

7.2.3.1.1.1 LoadKind

Description

The LoadKind defines the type of load to be performed.

Types and Exceptions

- `<<enumeration>> LoadKind (KERNEL_MODULE, DLL, DRIVER, SHARED_LIBRARY, EXECUABLE)`

7.2.3.1.1.2 DeploymentRequirement

Description

The DeploymentRequirement abstraction, as shown in Figure 7.35 on page 110, is used to provide the common definition for deployment requirements. The DeploymentRequirement is used to define the deployment requirement against a ServiceProperty.

Attributes

- `id: String`
  The id attribute indicates the identity of the ServiceProperty the deployment requirement is against.
- `value: ServiceProperty`
  The value attribute contains deployment requirement values that go against a ServiceProperty.

Constraints

The values for the value attribute must agree with the ServiceProperty definition as identified by the id attribute.

Semantics

The deployment requirement is used to specify the type of service needed and/or the capacity needed from a ServiceComponent. These deployment requirements are evaluated against the registered ServiceComponent's ServiceProperty(s) within a domain.

7.2.3.1.2 Artifacts Stereotypes

This section defines the stereotypes for artifacts. The artifacts stereotypes are depicted in Table 7.10 and Figure 7.32, which are extensions of the UML Artifact. The details of each artifact are described in the following subsections.

Table 7.10 - Artifacts Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
BITStream	Artifact	LoadableCode			Indicates an artifact that is object code for a prologic device (e.g., Field Programmable Gate Array device).
ComponentExecutable Code	Artifact	ExecutableCode		Minimum Executable parameters supported: identifier, naming context component reference, name bindingSee constraints in section below	Indicates an artifact that is an executable operating system main process that manifests either an SWRadioResource or/and ResourceFactory component a Component, which is registered with a naming service.
depends on	Association	N/A			Indicates an association that depicts a coding dependency where the code pointed needs to be loaded first.
Descriptor	Artifact	File			Indicates an artifact that is a deployment or component specification that conveys information on the element to be deployed.
DeviceConfiguration Descriptor	Artifact	Descriptor			Indicates a descriptor that describes the service components configuration for a node.
DeviceDescriptor	Artifact	Descriptor			Indicates a descriptor that describes a device.
DomainManager-Descriptor	Artifact	Descriptor			Indicates a descriptor that describes a domain manager component.
ExecutableCode	Artifact	LoadableCode	EntryPoint Name, Process Priority, stackSize	Executable parameters conform to argv of the POSIX exec family of functionsSee constraints in section below	Indicates an artifact that is an executable operating system main process or entry point.
executes	Association	N/A			Indicates an association that denotes execution code on a device

Table 7.10 - Artifacts Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
LibraryCode	Artifact	LoadableCode			Indicates an artifact that is loadable static or dynamic object code.
LoadableCode	Artifact	File			Indicates an artifact that defines the base code artifact definition and relationships for any type of component artifact.
loads	Association	N/A			Indicates an association that denotes the loading of code on a device
LogicalDevice ExecutableCode	Artifact	ServiceExecutable Code		Minimum Executable parameters supported: Identifier, Software Profile, Composite Device IORSee constraints in section below	Indicates an artifact that is an executable operating system main process that manifest a LogicalDevice component.
PropertiesDescriptor	Artifact	Descriptor			Indicates a descriptor that describes properties for a component definition or implementation.
ServiceExecutable Code	Artifact	ExecutableCode		Minimum Executable parameters supported: DeviceManager Registration Reference, Service NameSee constraints in section below	Indicates an artifact that is an executable operating system main process that manifests a Service.
SoftwareAssembly Descriptor	Artifact	Descriptor			Indicates a descriptor that describes an application assembly to be deployed.
SoftwareComponent Descriptor	Artifact	Descriptor			Indicates a descriptor that describes a component definition to be deployed.

Table 7.10 - Artifacts Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
SoftwarePkg Descriptor	Artifact	Descriptor			Indicates a descriptor that describes a component implementation to be deployed.
terminates	Association	N/A			Indicates an association that denotes termination executable code on a device
unloads	Association	N/A			Indicates an association that denotes the unloading of code from a device

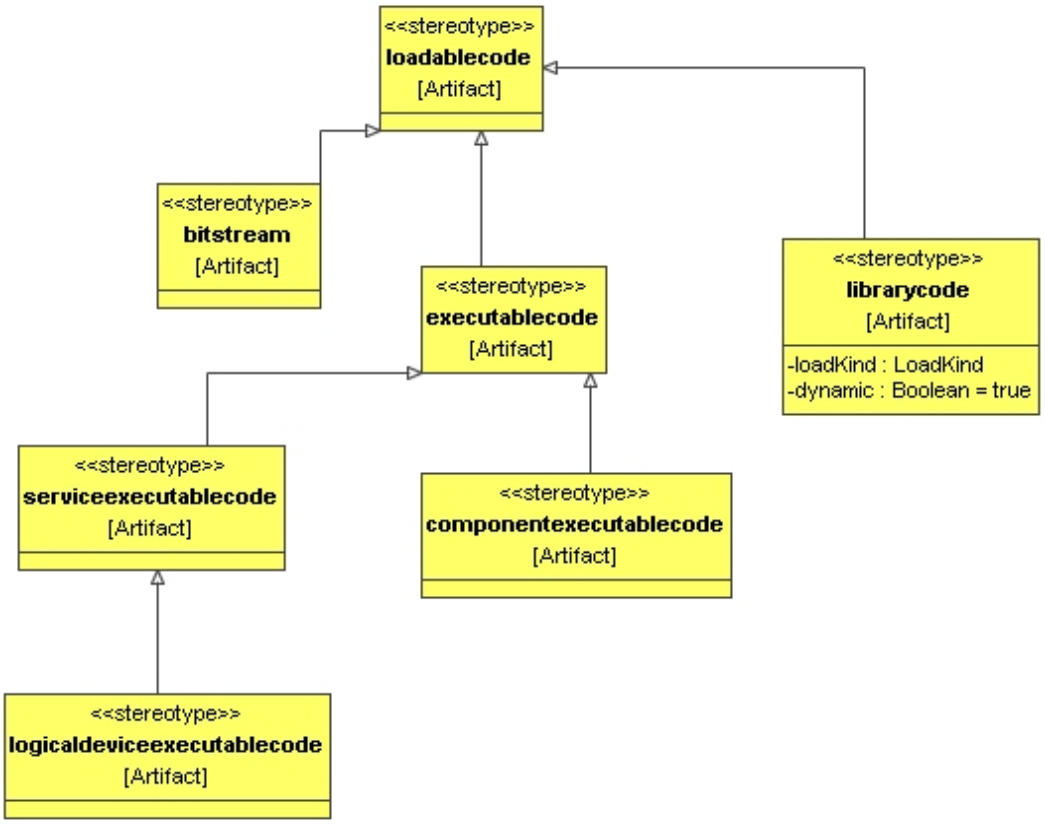

Figure 7.32 - Artifacts Relationships

7.2.3.1.2.1 ExecutableCode

Description

The ExecutableCode artifact, as shown in Figure 7.33, defines the definition and relationships that are common for main process types.

Attributes

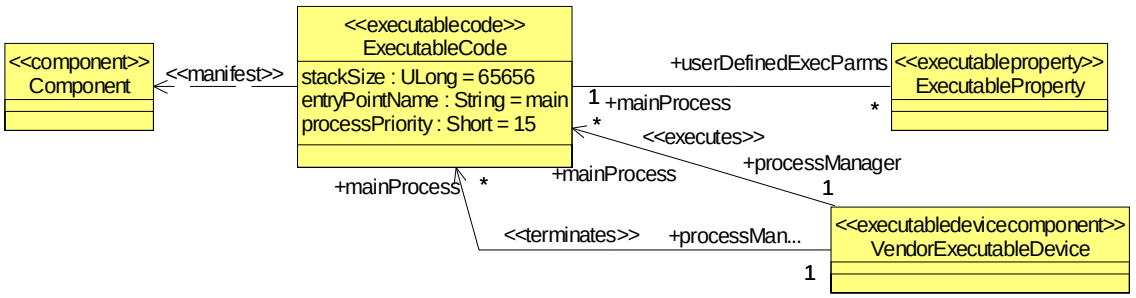

Figure 7.33 - ExecutableCode M1 Illustration

- `entryPointName: String = main`
  The entryPointName attribute indicates the name of the process to be created within the operating system.
- `processPriority: Ushort [0..1]`
  The processPriority attribute indicates the priority of the process to be created within the operating system.
- `stackSize: Ulong [0..1]`
  The stackSize attribute indicates the stack size of the process to be created within the operating system.

M1 Associations

- `userDefinedExecParms: ExecutableProperty [*]`
  For any main process or entry point, a user can specify executable parameters that are to be passed to the process upon creation.

Constraints

The ExecutableCode entry point's input parameters (id/value string pairs) shall conform to the standard argv of the POSIX exec family of functions, where argv(0) is the function name followed by id/value string pairs.

An ExecutableCode shall manifest a Component.

ExecutableCode properties shall be of type ExecutableProperty.

7.2.3.1.2.2 LibraryCode

Description

The LibraryCode artifact defines the LoadableCode artifact that is a loadable image.

Attributes

- `kind: LoadKind`
  The kind attribute indicates the type of load to be performed.
- `dynamic: Boolean = TRUE`
  The dynamic attribute indicates if the library to be loaded is dynamic or static. TRUE means dynamic loadable.

7.2.3.1.2.3 LogicalDeviceExecutableCode

Description

The LogicalDeviceExecutableCode artifact, a type of ServiceExecutableCode (see Figure 7.34), defines the operating system main process that manifests a LogicalComponent component, which is a specific type of ManagerServiceComponent.

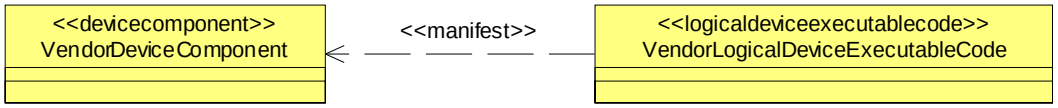

Figure 7.34 - LogicalDeviceExecutableCode M1 Illustration

Constraints

The LogicalDeviceExecutableCode shall accept the following executable parameters in addition to the ServiceExecutableCode parameters:

- Profile Name - The ID is “PROFILE_NAME” and the value is a string that is the full mounted file system file path name that is used for DeviceComponent profile attribute.
- Device Identifier - The ID is “DEVICE_ID” and the value is a string that is used for the Component’s identifier attribute.
- Composite Device Component Reference - The ID is “Composite_DEVICE_IOR” and the value is a string that is a DeviceCompositionComponent reference.

The LogicalDeviceExecutableCode Service Name executable parameter shall be used for the DeviceComponent’s label attribute value.

The LogicalDeviceExecutableCode shall add the DeviceComponent manifested by the process to the device composition using the Composite Device Component Reference executable parameter when specified.

The LogicalDeviceExecutableCode Device Identifier executable parameter shall be used for the DeviceComponent’s identifier attribute value.

The LogicalDeviceExecutableCode Profile Name executable parameter shall be used for the DeviceComponent’s softwareProfile attribute value.

A LogicalDeviceExecutableCode shall manifest a DeviceComponent.

7.2.3.1.2.4 LoadableCode

Description

The LoadableCode artifact, as shown in Figure 7.35, describes the definition and relationships that are common for all artifact types.

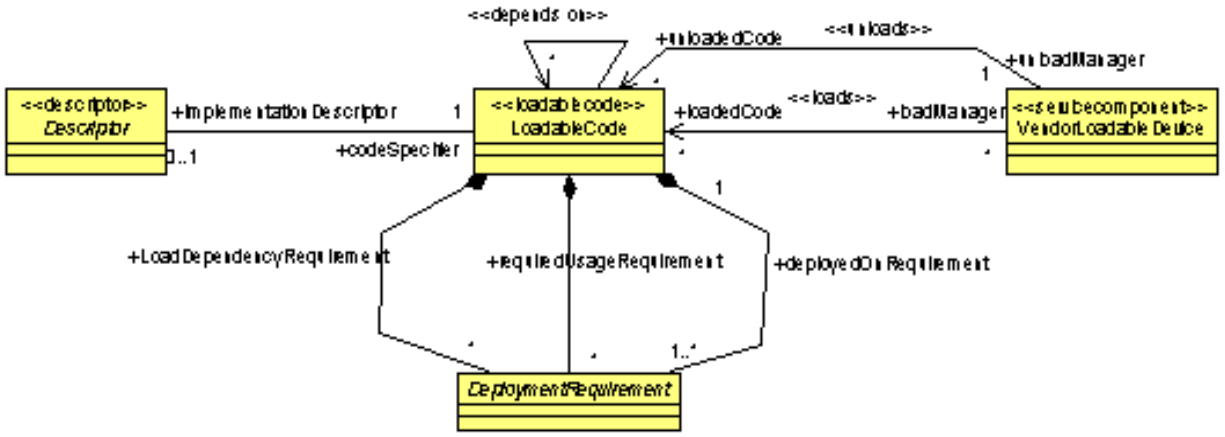

Figure 7.35 - LoadableCode M1 Illustration

M1 Attributes

- `compiler: DeploymentRequirementQualifier [0..1]`
  The compiler attribute indicates the compiler name and version used to create the loadable code.
- `deployedOnRequirement: DeploymentRequirement [1..*]`
  A LoadableCode artifact needs to specify “deployed on” deployment requirements in order for the artifact code gets deployed on the right communication channel’s device. The set of deployedOnRequirements specifies the device required and the capacity needed from the device.
- `loadDependencyRequirement: DeploymentRequirement [*]`
  A LoadableCode artifact can have dependency to other object code that requires loading and or execution first. The set of loadDependencyRequirements specifies the object code dependency.
- `requiredUsageRequirement: DeploymentRequirement [*]`
  A LoadableCode artifact requires usage of a service. The set of requiredUsageRequirements specifies the service required.

M1 Associations

- `implementationDescriptor: Descriptor [1]`
  LoadableCode artifacts are described by an implementation descriptor, which captures the deployment requirements and the type of artifact to be deployed

Semantics

A LoadableDevice manages the loading and unloading of LoadableCode on a loadable device.

7.2.3.1.2.5 ComponentExecutableCode

Description

The ComponentExecutableCode artifact, defines the operating system main process that manifests a Component and registers it to a naming service.

Constraints

ComponentExecutableCode shall accept the following three executable parameters:

- NamingContext Component Reference - The ID is “NAMING_CONTEXT_IOR” and the value is a stringified naming context reference. The reference is used to obtain the NamingContext path. The format of a NamingContext path is sequence of NamingContext where each “slash” (/) represents a separate naming context. A NamingContext is made up of ID and kind pair, which is indicated by “id.kind” in the NamingContext string. The NamingContext kind is optional and when not specified a null string will be used. For example, the structure of the naming context path is “/ SomeName / [optional naming context sequences].” There is at least one “slash” (/) in the Naming Context string.
- Name Binding - The ID is “NAME_BINDING” and the value is a string that corresponds to the name used to bind the manifested component to the naming service.
- Identifier - The ID is “COMPONENT_IDENTIFIER” and the value is a string that is used for the Component’s identifier value.

ComponentExecutableCode shall register the Component manifested by the process to the NamingService as specified by the NamingContext Component Reference executable parameter. The name binding registered to the NamingService shall be as specified by the Name Binding executable parameter.

The ComponentExecutableCode Identifier executable parameter shall be used for the manifested Component’s identifier attribute value.

7.2.3.1.2.6 ServiceExecutableCode

Description

The ServiceExecutableCode artifact, as shown in Figure 7.36, defines the operating system main process that manifests a Service component.

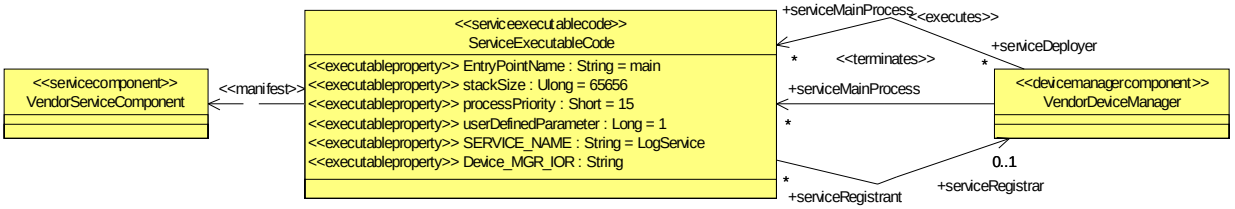

Figure 7.36 - ServiceExecutableCode M1 Illustration

Constraints

The ServiceExecutableCode shall accept the following user-defined executable parameters:

- DeviceManager Registration Reference - The ID is “DEVICE_MGR_IOR” and the value is a string that is the INodeRegistration component reference.

- Service Name - The ID is “SERVICE_NAME” and the value is a string that corresponds to the service name.

The ServiceExecutableCode shall register the ServiceComponent’s Service(s) manifested by the process to the DeviceManager using the DeviceManager Registration Reference executable parameter.

A ServiceExecutableCode shall manifest a ServiceComponent.

7.2.3.2 Applications Deployment

This section defines the interfaces (7.2.3.2.1) and components (7.2.3.2.2) that perform the deployment behavior within a platform.

7.2.3.2.1 Applications Deployment Interfaces

This section defines the interfaces that perform the deployment behavior within a platform. The deployments interfaces are Application and ApplicationFactory, which are described in the following subsections.

Types and Exceptions

- `DeviceAssignmentType( componentId: String, assignedDeviceId: String)`
  DeviceAssignmentType defines a structure that associates a component with the DeviceComponent upon which the component must execute.
- `DeviceAssignmentSequence`
  DeviceAssignmentSequence provides an unbounded sequence of DeviceAssignmentType.

7.2.3.2.1.1 Application

Description

The Application interface, as shown in Figure 7.37, provides for the control, configuration, and status of an instantiated application or waveform in the domain. The Application interface is specialization of the Resource interface that provides generic operations for controlling the deployed application.

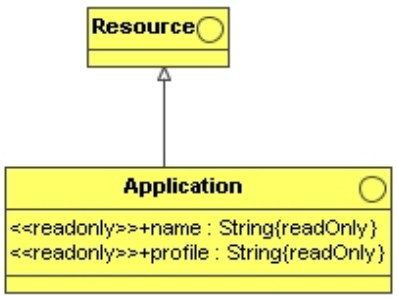

Figure 7.37 - Application Definition

Attributes

- `<<readonly>>name: String`
  The readonly name attribute contains the name of the created Application. The ApplicationFactory’s create operation name parameter provides the name content.

- `<<readonly>>profile: String`
  The readonly profile attribute contains the instantiated Application descriptor file reference.

Operations

- `getProvidedPorts (inout ports: PortSequenceType): {raises ( UnknownPorts )}`
  The getProvidedPorts operation returns object references only for input port names that match the external provided port names that are in the Application's component assembly descriptor. In the ports name/value pair sequence, each name corresponds to an external provided port name and each value corresponds to the object reference of the external provided port to be returned. The getProvidedPorts operation shall return all the external provided ports if the ports argument is zero size. The getProvidedPorts operation shall return only those provided ports specified in the ports argument if the ports argument is not zero size. The getProvidedPorts operation shall raise an UnknownPorts exception when one or more requested provided ports are invalid.
- `releaseObject(): (raises (ReleaseError)}`
  The releaseObject operation terminates execution of the ApplicationManager.

7.2.3.2.1.2 ApplicationDeploymentData

Description

The ApplicationDeploymentData is an interface, as shown in Figure 7.39, that provides detail information on Application instance deployment.

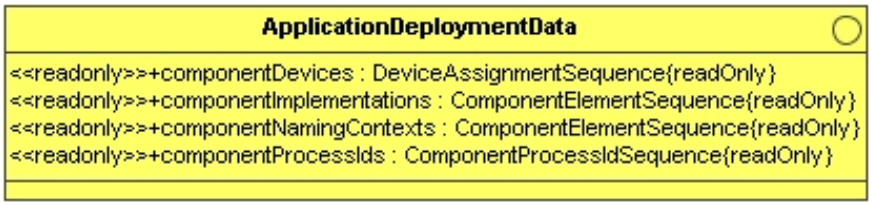

Figure 7.38 - ApplicationDeploymentData Definition

Attributes

- `<<readonly>>componentDevices: DeviceAssignmentSequence`
  The readonly componentDevices attribute contains a list of DeviceComponents, which each ApplicationAssembly's component uses, is loaded on, or is executed on.
- `<<readonly>>componentImplementations: ComponentElementSequence`
  The readonly componentImplementations attribute contains the list of components' implementation IDs within the Application for those components created.
- `<<readonly>>componentNamingContexts: ComponentElementSequence`
  The readonly componentNamingContexts attribute contains the list of components' Naming Context using Naming Service.
- `<<readonly>>componentProcessIds: ComponentProcessIdSequence`
  The readonly componentProcessIds attribute contains the list of components' process IDs within the Application for components that are manifested within an ExecutableCode on an ExecutableDeviceComponent.

Types and Exceptions

- `ComponentProcessIdType( componentId: String, processId: long)`
  The ComponentProcessIdType defines a type for associating a component with its process ID. This type can be used to retrieve a process ID for a specific component.
- `ComponentProcessIdSequence`
  The ComponentProcessIdSequence type defines an unbounded sequence of componentProcess IdTypes.
- `ComponentElementType( componentId:String, elementId: String)`
  The ComponentElementType defines a type for associating a component with an element (e.g., naming context, implementation ID).
- `ComponentElementSequence`
  The ComponentElementSequence defines an unbounded sequence of ComponentElementType.

7.2.3.2.1.3 ApplicationFactory

Description

The ApplicationFactory interface, as shown in Figure 7.39, provides the dynamic mechanism to create instances of a specific type of Application (e.g., waveform) in the domain.

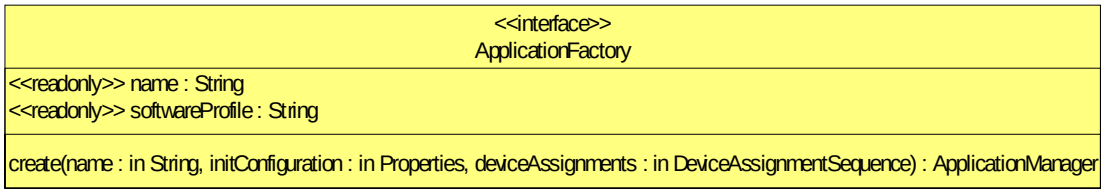

Figure 7.39 - ApplicationFactory Definition

Attributes

- `<<characteristic>>capabilityManager: Boolean = False`
  The characteristic capabilityManager attribute indicates the ApplicationFactory behavior in regards to CapacityProperty(s). A value of True means the ApplicationManager manages CapacityProperty(s), otherwise it does not.
- `<<readonly>>name:`
  The readonly name attribute contains the name of the installed Application.
- `<<readonly>>softwareProfile:`
  The readonly softwareProfile attribute contains the installed Application description file reference.

Operations

- `create(in name: String, in initConfiguration: Properties, in deviceAssignments: DeviceAssignmentSequence, return ApplicationManager):{raises ( CreateApplicationError, CreateApplicationRequestError, InvalidInitConfiguration )}`

The create operation provides the capability of creating an ApplicationManager instance.

Types and Exceptions

- `<<exception>>CreateApplicationRequestError`
  This exception is raised when the DeviceAssignmentSequence contains 1 or more invalid Application component-to-device assignment(s).
- `<<exception>>CreateApplicationError`
  The CreateApplicationError exception, specialization of SystemException, is raised when the create operation is unsuccessful due to internal processing errors. The error number indicates an ErrorNumberType value (e.g., E2BIG, ENAMETOOLONG, ENFILE, ENODEV, ENOENT, ENOEXEC, ENOMEM, ENOTDIR, ENXIO, EPERM). The message is component-dependent, providing additional information describing the reason for the error.
- `<<exception>> InvalidInitConfiguration`
  The InvalidInitConfiguration exception is raised when the input initConfiguration parameter is invalid.

Semantics

The create operation provides the capability of deploying an instance of the Application by making dynamic decisions on which DeviceComponent(s) the Application's components are deployed on and which Service(s) are used by an Application.

The create operation determines which registered domain Services can satisfy the Application's deployment requirements as stated in the installed Application descriptor. The create operation also provides the mechanism to direct which DeviceComponent(s) are to be used for Application deployment instead of the ApplicationFactoryComponent making the DeviceComponent decision.

The create operation shall use descriptor information as referenced by the softwareProfile to determine the Application deployment requirements.

If input deviceAssignments (not zero length) are provided, the create operation verifies each device assignment for the specified component against the Application's deployment requirements.

The create operation shall allocate the Application's deployment requirements against candidate Service(s) to determine which candidate Service(s) satisfy all the Application's deployment requirements and partitioning requirements (e.g., components HostCollocation, components process thread collocation, etc.). The create operation shall only use Service(s) whose capability and capacity characteristics (expressed in the Service(s)' ServiceProperties) satisfy the allocation requirements (i.e., deployment and partitioning requirements) specified for the Application components. The create operation shall use the ServiceProperty's CapabilityModel or delegate as specified by Service's ServiceProperty's capabilityModel and locallyManaged attributes for determining Service(s) that can satisfy an Application's deployment requirement. The actual Service(s) chosen will reflect changes in capacity based upon component deployment requirements allocated to them, which may also cause state changes for the Service(s).

The create operation shall load the Application components (including all of the Application-dependent components) to the chosen DeviceComponent(s).

The create operation shall execute the Application components (including all of the application-dependent components) as specified in the Application's descriptor. The create operation shall use each component's implementation stack size and priority attributes, when specified, for the ExecutableDevice's execute options parameters.

The create operation shall pass the mandatory execute parameters as specified for ResourceExecutableCode for each Application component's implementation that has an entry point.

The create operation shall pass ResourceExecutableCode's executable parameters (NamingContext, Name Binding, Identifier) to an ExecutableDevice's execute formal parameter named parameters. The create operation creates any naming contexts that do not exist to which the component will bind to. The structure of the naming context path shall be "/ DomainName / [optional naming context sequences]." In the naming context path, each "slash" (/) represents a separate naming context.

The ResourceExecutableCode's executable Name Binding parameter value shall be set to a string in the format of "ComponentName_UniqueIdentifier." The ComponentName value shall be the component instantiation namingservice's name attribute in the Application's component assembly descriptor. The UniqueIdentifier is determined by the implementation.

The create operation uses "ComponentName_UniqueIdentifier" to retrieve the component's object reference from the Naming Context. Due to the dynamics of bind and resolve to NamingService, the create operation should provide sufficient attempts to retrieve component object references from NamingService prior to generating an exception.

The ResourceExecutableCode's executable Identifier parameter value shall be set to a string in the format of "Component_Instantiation_Identifier: Application_Name." The Component_Instantiation_Identifier shall be the component's instantiation id attribute in the Application's component assembly descriptor. The Application_Name field shall be identical to the create operation's input name parameter. The Application_Name field provides a specific instance qualifier for executed ResourceComponent(s).

The create operation shall pass a component's implementation's ExecutableProperty(s) to an ExecutableDevice's execute formal parameter named parameters. The create operation shall pass Executable Property values as string values.

The create operation shall, in order, initialize application components that support the LifeCycle interface, then establish connections for application components that support the PortConnector interface, and finally configure the application components that support the PropertySet interface and have ConfigureProperty(s) described in the application's component assembly descriptor. The Application's assemblyController shall be configured after the configuration of all application components.

The create operation uses the PortSupplier interface for obtaining provider interfaces for a connection.

The create operation input initConfiguration properties shall only apply to the assemblycontroller component of the deployed Application as defined in the Application's descriptor. A deployed component's configuration property values shall only come from the Application's descriptor, not from the component's implementation or component definition descriptors.

The create operation configures an Application's assemblyController component provided the assemblyController has ConfigureProperty(s). The create operation shall use the union of the input initConfiguration properties of the create operation and the assemblyController's ConfigureProperties. The input initConfiguration parameters shall have precedence over the assemblyController's configure property values.

The create operation shall, when creating a ResourceComponent from a ResourceFactory, pass the associated ResourceFactory's ConfigureProperty(s) as qualifiers parameters to the referenced ResourceFactory component's createResource operation.

The create operation shall interconnect ResourceComponent(s) (Application's components or DeviceComponents') ports in accordance with the Application's component assembly descriptor. The create operation obtains provider ports in accordance with the Application's component assembly via PortSupplier's getProvidedPorts operation.

The connections to domain Service(s) such as LogService, FileManager, FileSystem, Event Service, and NamingService are specified as component's connections using domainfinder in the Application's component assembly descriptor. Domain Service(s) are services that have been registered to the domain.

For connections established for an EventService's event channel, the create operation shall connect a PushConsumer or PushSupplier object to the event channel as specified in the component's connection in the Application's component assembly descriptor. If the event channel does not exist, the create operation shall create the event channel.

The create operation establishes connections to ResourceComponent(s) using the PortConnector::connectPort operation. The create operation shall use the connection id attribute as the unique identifier for a specific connection when provided in the Application's component assembly descriptor. The create operation shall create a connection ID when no connection id is specified for a connection in the Application's component assembly descriptor.

If the Application is successfully created, the create operation returns an ApplicationManager component reference for the created Application. A sequence of created Application references can be obtained using the DomainManagerComponent's readonly applications attribute (getApplications operation). No additional semantics.

The create operation shall raise the CreateApplicationRequestError exception when the DeviceAssignmentSequence parameter contains one (1) or more invalid Application component to device assignment(s).

The create operation shall raise the CreateApplicationError exception when the Application cannot be successfully instantiated due to internal processing error(s).

The create operation shall raise the InvalidInitConfiguration exception when the input initConfiguration parameter is invalid. The InvalidInitConfiguration invalidProperties identifies the properties that are invalid.

The created ApplicationManager's name attribute shall be identical to the input name parameter.

7.2.3.2.2 Application Deployment Stereotypes

This section defines the components that perform the deployment behavior within a platform. The component stereotypes are depicted in the Table 7.11, which are extensions of the UML Component. The details of each component are described in the following subsections.

Table 7.11 - Applications Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
ApplicationManager	Component	Resource Component		See constraints in section below	Represents the proxy for deployed Application instance.
ApplicationFactory Component	Component	Component	capability Manager	See constraints in section below	Represents the deployment machinery for Application's Descriptor.

7.2.3.2.2.1 ApplicationManager

Description

The ApplicationManager is a type of ResourceComponent as shown in Figure 7.40. The ApplicationManager provides the interface for the control, configuration, and status of an instantiated application or waveform in the domain. The ApplicationManager is the proxy for the deployed Application component instance.

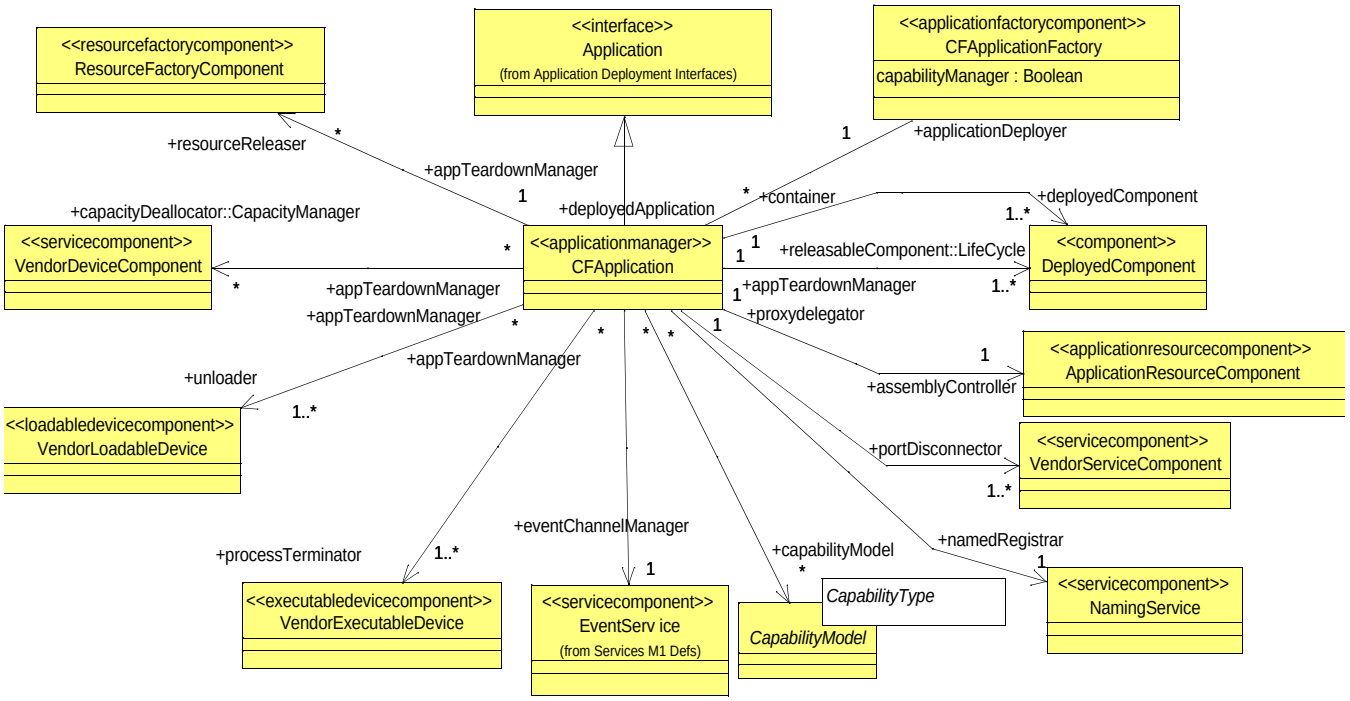

Figure 7.40 - ApplicationManager Definition

M1 Associations

- `assemblyController: ApplicationResourceComponent [1]`
  The Application's assemblyController that ApplicationManager delegates operations to.
- `capacityDeallocator: VendorDeviceComponent [*]`
  The DeviceComponents that were allocated capacities during the Application deployment are used to return capacities back to these devices.
- `capabilityModel: CapabilityModel [*]`
  The capabilityModels for ServiceProperty(s) used and managed by the Application.
- `deployedComponent: DeployedComponent [1..*]`
  The Application instance's deployed Components that are deployed.
- `eventChannelManager: EventService [1]`
  The EventService that manages the creation and destruction of EventChannels.
- `portDisonnector: VendorServiceComponent [1..*]`
  The ServiceComponent(s) provide the capability to disconnect provided ports from their required ports.
- `namedRegistrar: NamingService [1]`
  The NamingService that has the deployed Application components references.
- `releasableComponent: DeployedComponent [1..*]`
  The deployed Component(s) that were manifested during the Application deployment are released.

- `resourceReleaser: ResourceFactoryComponent [*]`
  The ResourceFactory(s) that were used during the Application deployment are used to release ApplicationResourceComponent(s) from these ResourceFactory(s).
- `processTerminator: VendorExecutableDevice [1..*]`
  The ExecutableDevices that executed ExecutabeCode during Application deployment are used to terminate these processes.
- `unloader: VendorLoadableDevice [1..*]`
  The LoadableDevices that were loaded with ObjectCode during the Application deployment are used to unload the ObjectCode from these devices.

Constraints

The ApplicationManager shall be either associated with DeviceComponent(s) or CapabilityModel(s) for the deallocate capacity behavior.

The ApplicationManager shall realize the Application interface.

Semantics

The ApplicationManager shall delegate the implementation of the inherited ResourceComponent operations (runTest, start, stop, configure, and query) to the Application instance's assembly controller (e.g., ApplicationResourceComponent). The ApplicationManager shall propagate exceptions raised by these operations. The initialize operation is not propagated to the Application's assembly controller. The initialize operation causes no action within an ApplicationManager.

The ApplicationManager releaseObject operation shall deallocate Application's required capacities against the ServiceComponent(s) that were obtained from or associated with. The actual DeviceComponent(s) may reflect changes in capacity based upon component capacity requirements deallocated from them, which may also cause state changes for the DeviceComponent(s). The releaseObject operation shall release all references to the Application component instances. The releaseObject operation shall disconnect port connections to non-application component providers that have been connected based upon the Application's component assembly descriptor. The releaseObject operation shall disconnect the Application instance's components consumers and producers from an EventService's event channel. The releaseObject operation may destroy an EventService's event channel when no more consumers and producers are connected to it. For components (e.g., Resource, ResourceFactory) that are registered with Naming Service, the releaseObject operation unbinds those components and destroys the associated naming contexts as necessary from the NamingService. The releaseObject operation shall disconnect ports first, then release the Application instance's ResourceComponent(s) and ResourceFactory(s), terminate the Application instance's ExecutableCode, and lastly unload the Application instance's LoadableCode from the DeviceComponent(s). The releaseObject operation shall raise a ReleaseError exception when the releaseObject operation unsuccessfully releases the Application instance due to internal processing errors.

An ApplicationManager implementation could realize the ApplicationDeploymentData interface to provide details on the deployment on the Application instance.

7.2.3.2.2.2 ApplicationFactoryComponent

Description

The ApplicationFactoryComponent provides the dynamic mechanism to create instances of a specific type of Application (e.g., waveform) in the domain. Figure 7.41 depicts the ApplicationFactory's capacity constraints and Figure 7.42 depicts the relationships that are common for all ApplicationFactory(s).

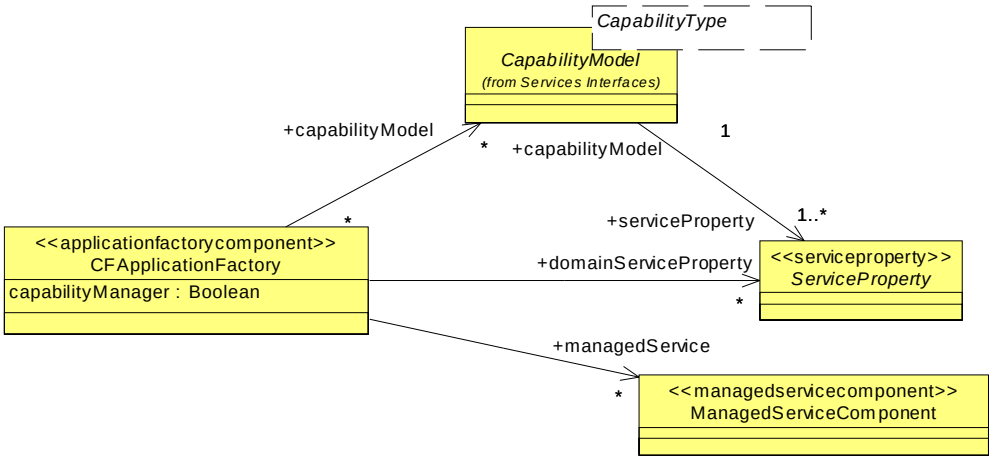

Figure 7.41 – ApplicationFactory Capability Manager Relationships

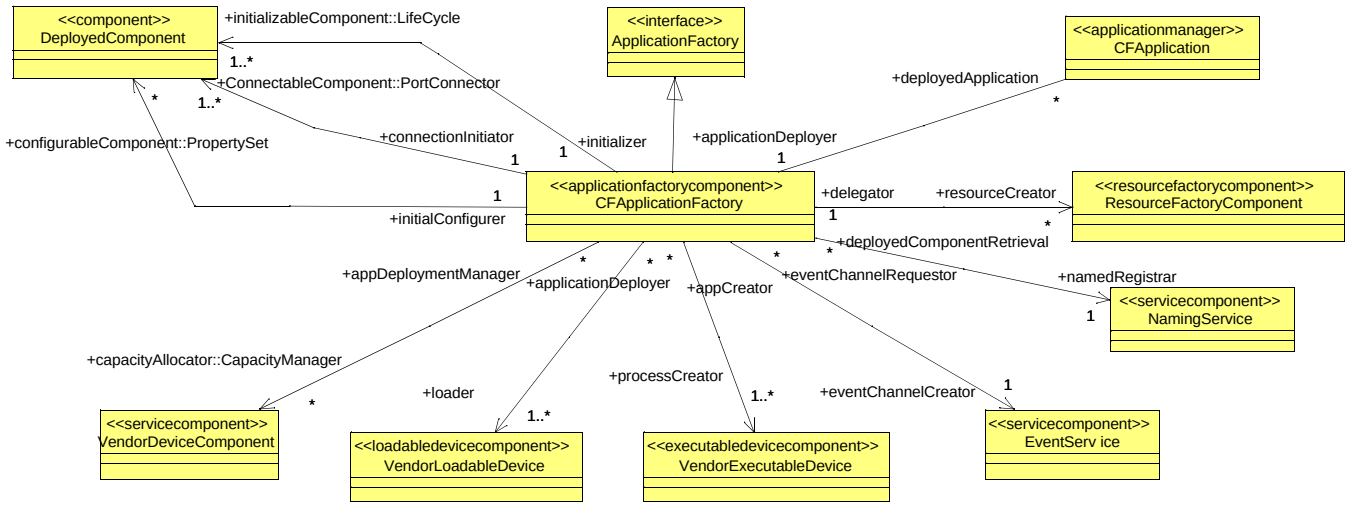

Figure 7.42 - ApplicationFactory M1 Illustration

Attributes

- `capabilityManager: Boolean = False`
  The capabilityManager attribute indicates the ApplicationFactoryComponent behavior in regards to CapacityProperty(s). A value of True means the ApplicationManagerComponent manages CapacityProperty(s), otherwise it does not.

M1 Associations

- `capabilityModel: CapabilityModel [*]`
  The CapabilityModel(s) used for determining which domainServiceProperty(s) can satisfy the Application's deployment requirements.
- `configurableComponent: DeployedComponent [*]`
  The deployed components that are configurable by the ApplicationFactoryComponent.
- `deployedApplication: CFApplicationManager [*]`
  The ApplicationManager that manages the instantiated Application.
- `domainServiceProperty: ServiceProperty [1..*]`
  The registered Services' ServiceProperty(s) that are used for determining Services to be used for Application deployment.
- `eventChannelCreator: EventService [1]`
  The EventService that manages EventChannels.
- `initializableComponent: DeployedComponent [*]`
  The deployed Component(s) that are initializable (LifeCycle interface) by the ApplicationFactory component.
- `loader: VendorLoadableDevice [1..*]`
  The LoadableDeviceComponent(s) are used to load ObjectCode during the Application deployment.
- `managedService: ManagedServiceComponent [*]`
  A managedService manages its own CapacityModel(s).
- `NamedRegistrar: NamingService[1]`
  The NamingService that contains deployed component object references.
- `portConnector: DeployedComponent [1..*]`
  The deployed Component(s) that are connected to Services and to other Component(s) by the ApplicationFactory component.
- `processCreator: VendorExecutableDevice [1..*]`
  The ExecutableDeviceComponent(s) are used to execute Application main processes during Application deployment.
- `resourceCreator: ResourceFactoryComponent [*]`
  The ResourceFactory can be used during the Application deployment as an optional means of creating ApplicationResourceComponent(s).

Constraints

The identifier attribute shall be identical to the installed Application's descriptor id attribute.

The name attribute shall be identical to the installed Application's descriptor name attribute.

When capabilityManager attribute is False the ApplicationFactoryComponent shall only use ServiceProperty(s) that are locally managed by a ServiceComponent, otherwise the ApplicationFactory manages the ServiceProperty(s). Characteristic properties can be either managed or unmanaged by the ApplicationFactoryComponent.

The ApplicationFactoryComponent shall realize the ApplicationFactory interface.

When the capabilityManager attribute is true, then the ApplicationFactory Component shall support the ServiceProperty's capabilityModel for CapacityProperty.

The ApplicationFactoryComponent shall support the ServiceProperty's capabilityModels for CharacteristicProperty, CharacteristicSelectionProperty, and CharacteristicSetProperty. Characteristic properties can be either managed or unmanaged by a ServiceComponent.

The ApplicationFactoryComponent shall associate only one assemblyController to each Application instance.

8 Platform Specific Model (PSM)

The PSM consists of CORBA and XML that are based upon the UML Profile for Component Framework. The PIM to PSM transformation rules are not universal rules for creating *any* PSM, but only used for the purpose of this specification. This section defines a non-normative reference PSM. Non-CORBA PSMs may also be fully compliant to this specification as a whole.

The rule set for transforming UML packages, interfaces, types, and exceptions into CORBA constructs are as follows:

1. UML interfaces and interface extensions map to CORBA interfaces. The CORBA interface names are without the prefix “I” in the interface name as used in the UML profile for Component Framework and in the PIM Facilities.
2. UML attributes with readonly and readwrite map to CORBA attributes in CORBA interfaces.
3. UML attributes with configureproperty, queryproperty and testproperty do not map to CORBA attributes in CORBA interfaces. Instead XML definitions are used that follow the Property types as defined in UML Profile for Component Framework::Application and Device Components::Properties section.
4. UML classes without operations that are not stereotyped and used for type definitions map to CORBAStruct stereotypes in the CORBA interfaces and modules. The parent classes do not get translated into CORBA types instead the parent class attributes are added to the subclass in the CORBA definition.
5. UML <<datatype>> maps to CORBA basic types. Primitive types are mapped to CORBA primitive types and primitive sequence types are mapped to CORBA Typedef of primitive sequence types.
6. UML exceptions and exception extensions map to CORBA exceptions. There is no specializations of exceptions in CORBA so the (UML Profile for Component Framework::Application and Device Components::BaseTypes) SystemException definition does not appear in the generated CORBA interfaces but all the specialization exceptions of SystemException are in the CORBA interfaces with the same attributes as defined for SystemException.
7. UML attributes that have a cardinality of many [*] map to a CORBA Typedef of sequence types.
8. UML operations map to operations in the CORBA interfaces.
9. Transformations are only performed for concrete classes, not for template classes. Concrete classes that bind to template classes are used in the PSM.
10. For Interfaces that reference a component stereotype for a type, the “component” qualifier is removed from the name. For Example, FileManagerComponent would become FileManager as the type for the parameter or attribute.
11. UML attributes with constant stereotype map to CORBA constants in CORBA interfaces.
12. Basic types (e.g., Any, Object) map to CORBA types.

The CORBA PSM corresponds to the UML Profile for Component Framework that maps to a CORBA module named CF (Component Framework). The CF CORBA module is an existing IDL definition used in industry, therefore the CF IDL does not follow all of the OMG CORBA guidelines (e.g., operation, attribute, and parameter names), in order to reduce impact on industry. The CF CORBA module is broken up into multiple files as shown in Table 8.1. The reason for the break-up into multiple files is a memory foot print size for the embedded environment. Specific interfaces are only used

and implemented on certain devices within the platform. The interfaces used vary by the type of developers (waveform, device, domain management) for a platform. These developers use different sets of interfaces for the components they are developing.

Table 8.1 - Component Framework CORBA Module Overview

UML Profile for Component Framework Specification Sections			IDL File	CORBA Module
Application and Device Components		BaseTypes	CFCommonTypes.idl	CF
			CFBaseTypes.idl	CF
		BaseTypes - each primitive sequence type is mapped to its own file	CFPortTypes_BooleanSequence.idl CFPortTypes_CharSequence.idl CFPortTypes_ShortSequence.idl CFPortTypes_UshortSequence.idl CFPortTypes_LongSequence.idl CFPortTypes_UlongSequence.idl CFPortTypes_LongLongSequence.idl CFPortTypes_UlongLongSequence.idl CFPortTypes_FloatSequence.idl CFPortTypes_DoubleSequence.idl CFPortTypes_LongDoubleSequence.idl CFPortTypes_WcharSequence.idl CFPortTypes_WstringSequence.idl CFPortTypes.idl	PortTypes within CF
		Resource Components Interfaces except for ResourceFactory	CFResources.idl	CF
		Resource Components Interfaces - ResourceFactory only	CFResourceFactory. idl	CF
		Device Components Interfaces	CFDevices.idl	CF
Infrastructure	Domain Management	Device Management Interfaces - Each interface maps to its own IDL file.	CFServiceRegistration.idl, CFDeviceManager.idl	CF
		Domain Management Interfaces - Each interface maps to its own IDL file	CFDomainEventChannels.idl, CFDomainInstallation.idl, CFDeviceManagerRegistration.idl, CFDomainManager.idl	CF
		Domain Event Channels	CF_SE_DomainEvent.idl CF_SE_StateEvent.idl	CF
	Services	File Services Interfaces - Each interface maps to its own IDL file	CFFile.idl, CFFileSystem.idl, CFFileManager.idl	CF
		Services Interfaces	CFStateManagement.idl	CF
	Deployment	Applications Deployment Interfaces	CFApplications.idl	CF

Other non-CORBA PSM transforms (e.g., XML) are as follows:

1. The UML Profile for Component Framework::Application and Device Components::Properties maps to Properties XML definitions. Each property definition maps to an XML element definition. Abstract classes are not directly transformed into XML, instead their attributes are used for the concrete subclass XML definitions. Attributes with default values are created as XML attributes. All other attributes are created as XML elements. The name and ID is a unique value for each property in an XML properties set. Only the properties attributes stated in the Properties section are used for the XML properties definition. Specific transformations of the properties are as follows:

    - Primitive types are mapped to the corresponding enumeration literal in the SimpleType XML element.
    - EnumerationProperty attributes map to the EnumerationLiteral XML element. The attribute name and value maps to the label and value xml elements.
    - ConfigureProperty and QueryProperty that are primitive types maps to the XML ConfigureQuerySimpleProperty XML element.
    - ConfigureProperty and QueryProperty that are primitive sequence types maps to the XML ConfigureQuerySimpleSeqProperty XML element.
    - ConfigureProperty and QueryProperty that are a StructProperty type maps to the XML ConfigureQueryStructProperty XML element.
    - ConfigureProperty and QueryProperty that are a StructProperty sequence type maps to the XML StructSequenceProperty XML element.
    - TestProperty maps to the XML TestProperty element
    - CharacteristicProperty maps to XML CharacteristicProperty
    - CapacityProperty mapes to XML CapacityProperty
    - CharacteristicSelectionProperty maps to XML CharacteristicSelectionProperty
    - CharacteristicSetProperty maps to XML CharacteristicSetProperty
    - ExecutableProperty maps to XML ExecutableProperty

2. In industry there are two sets of XML definitions that could be used for the deployment of components with a platform, which are the Document Type Definitions (DTDs) as described in reference 3.2.1.1 and CCM Schema XML as described in the COBRA Components Model (CCM). The relationships of these XML elements to the CF components are depicted in Table 8.2

.

Table 8.2

Component Type	Descriptors PSM	
	Document Type Definitions XML	**CCM Schema XML**
ApplicationManager ApplicationFactoryComponent	Software Assembly Descriptor, Software Package Descriptor, Software Component Descriptor, Properties Descriptor	ComponentPackageDescription, ComponentInterfaceDescription, ComponentAssemblyDescription, MonolithicImplementationDescription, and Properties XML
DeviceManagerComponent	Device Configuration Descriptor	
DomainManagerComponent	Domain Configuration Descriptor	
Component ServiceComponent	Software Assembly Descriptor, Software Package Descriptor, Software Component Descriptor, Properties Descriptor	ComponentPackageDescription, ComponentInterfaceDescription, ComponentAssemblyDescription, MonolithicImplementationDescription, and Properties XML

Annex A Software Radio Reference Sheet

The Software Radio specification responds to the requirements set by “Request for Proposals for a Platform Independent Model (PIM) and CORBA Platform Specific Model (PSM)” (swradio/02-06-02). The original specification (dtc/05-10-02) has been reorganized into 5 volumes, as follows:

Volume 1. Communication Channel and Equipment

This specification describes a UML profile for communication channel. The profile provides definitions for creating communication channel and communication equipment definitions. The specification also provides radio control facilities and physical layer facilities PIM for defining interfaces and components for managing communication channels and equipment for a radio set or radio system. Along with the profile and facilities is a platform specific model transformation rule set for transforming the communication channel into an XML representation and CORBA interfaces for the radio control facilities.

Volume 2. Component Document Type Definitions

This specification defines the content of a standard set of Data Type Definition (DTD) files for applications, components, and domain and device management. The complete DTD set is contained in Section 7, Document Type Definitions. XML files that are compliant with these DTD files will contain information about the service components to be started up when a platform is power on and information for deploying installed applications.

Volume 3. Component Framework

This specification describes a UML profile for component framework. The profile provides definitions for applications, components (properties, ports, interfaces, etc.), services, artifacts, logical devices, and infrastructure domain management components. In the profile are also library packages that contain interfaces for application, service, logical device, and infrastructure domain management components. Along with the profile is a platform specific model transformation rule set for transforming the profile model library interfaces into CORBA interfaces.

Volume 4. Common and Data Link Layer Facilities

This specification describes a set of facilities PIM for application and component definitions. The set of facilities are common layer facilities and data link layer facilities that can be utilized in developing waveforms and platform components, which promote the portability of waveforms across Software Defined Radios (SDR).

Volume 5. POSIX Profiles

This specification defines the application environment profiles for embedded constraint systems, based on Standardized Application Environment Profile - POSIX® Realtime Application Support (AEP), IEEE Std 1003.13-1998.

INDEX

A
Acknowledgements 8
Any 13
Application 62
Application Components 60
Application interface 112
ApplicationDeploymentData 113
ApplicationFactory 114
ApplicationFactoryComponent 120
ApplicationManager 117
ApplicationResourceComponent 63
Applications and Devices package 11
Applications Deployment 112
Artifacts 103
Artifacts Stereotypes 105

B
Base Types 13
BooleanSequence 13

C
CapabilityModel 70
CapacityManager 48
CapacityModel 71
CapacityProperty 24
Character 13
CharacteristicModel 71
CharacteristicProperty 25
CharacteristicSelectionProperty 26
CharacteristicSetProperty 26
CharSequence 14
Component 44
ComponentExecutableCode 111
ComponentIdentifier 36
ComponentInstantiation 63
ConfigureProperty 27
Conformance 1
ControllableComponent 36

D
Definitions 5
Deployment 103
DeploymentRequirement 104
Device 49
Device Components 47
Device Management 96
Device management stereotypes 98
DeviceComponent 55
DeviceComposition 50
DeviceCompositionComponent 57
DeviceDriver 57
DeviceManager interface 96
DeviceManagerComponent 98
DeviceManagerRegistration 90
Domain Event Channels 101
Domain management 87
Domain management stereotypes 93
DomainEventChannels 88
DomainInstallation 88
DomainManager interface 92
DomainManagerComponent 93
Double 14
DoubleSequence 14

E
EnumerationProperty 28
ErrorNumberType 14
ExecutableCode 108
ExecutableDevice 50
ExecutableDeviceComponent 58
ExecutableProperty 28

F
File interface 80
File Services 78
File Services Interfaces 79
File Services stereotypes 86
FileManager 81
FileManagerComponent 86
FileSystem interface 83
FileSystemComponent 87
Float 15
FloatSequence 15

H
horizontal communication scenario 69

I
Infrastructure 70
InputValueProperty 29
Interface and Port stereotypes 33
InvalidFileName 15
InvalidObjectReference 15
issues/problems vi

L
LibraryCode 108
LifeCycle 37
LinkLayerControlResource 64
Literal specifications 20
LoadableCode 110
LoadableDevice 53
LoadableDeviceComponent 59
LogicalDeviceExecutableCode 109
Long 15
LongDouble 15
LongDoubleSequence 16
LongLong 16
LongLongSequence 16
LongSequence 16

M
ManagedServiceComponent 75
MediumAccessControlResource 66

N
NameVersionCharacteristic 16
NetworkLayerResource 67

Non-normative References 4
Normative References 3

O
Object 16
Object Management Group, Inc. (OMG) v
ObjectReference 17
ObjectRefSequence 17
Octet 17
OctetSequence 17
OMG specifications v

P
PhysicalLayerResource 67
Platform Independent Model (PIM) 1
Platform management 87
Platform Specific Model (PSM) 1, 123
PortConnector 37
PortSupplier 38
Properties 17, 22
Property 29
PropertySet 39
PropertyValue 17

Q
QueryProperty 29

R
References 3
RequiresPort 34
Resource Components 35
Resource Components Stereotypes 44
Resource interface 40
ResourceComponent 45
ResourceFactory 46
ResourceFactory interface 42

S
Scope 1
Service stereotypes 74
ServiceComponent 78
ServiceComponentInstantiation 68
ServiceConnector 69
ServiceExecutableCode 111
ServiceProperty 30
ServiceRegistration interface 98
Short 18
ShortSequence 18
SimpleProperty 31
StateManagement 72
StreamPort 34
StringSequence 18
StructProperty 32
Symbols 8
SystemException 18

T
Terms and definitions 5
TestableObject interface 43
TestDefProperty 32
TestProperty 33
TimeType 20
typographical conventions vi

U
ULong 18
ULongLong 19
ULongLongSequence 19
ULongSequence 19
UShort 19
UShortSequence 19

V
vertical communication scenario 69

W
WaveformLayerResource 69
WChar 19
WCharSequence 20
WString 20
WStringSequence 20